

AD-A172 379

A DECISION SUPPORT SYSTEM FOR SPACE TECHNOLOGY

1/2

TRADEOFFS: A MICROCOMPUTER. (U) AIR FORCE INST OF TECH

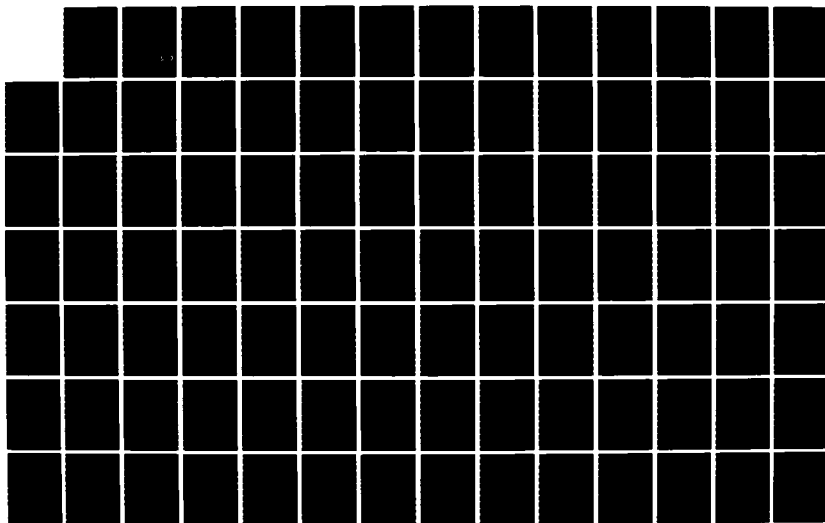
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. B G SCHINELLI

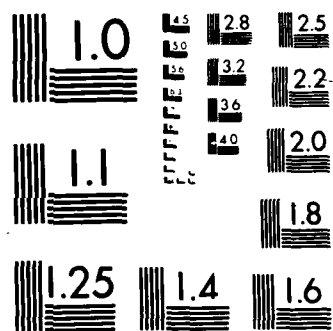
UNCLASSIFIED

13 DEC 85 AFIT/GOR/OS/85D-17

F/G 12/2

NL





AFIT/GOR/OS/85D-17

AD-A172 379

A DECISION SUPPORT SYSTEM FOR
SPACE TECHNOLOGY TRADEOFFS:
A MICROCOMPUTER APPLICATION

THESIS

AFIT/GOR/OS/85D-17

Bruce G. Schinelli, B.S.
1st Lt., USAF

DTIC FILE COPY

DTIC
ELECTE
OCT 02 1986
S D
E

Approved for public release: distribution unlimited

86 10 2 166

A DECISION SUPPORT SYSTEM FOR SPACE TECHNOLOGY
TRADEOFFS: A MICROCOMPUTER APPLICATION

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Bruce G. Schinelli, B.S.

1st Lt., USAF

December 1985

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	

Approved for public release: distribution unlimited



Preface

The purpose of this study was to implement a Decision Support System on a microcomputer to help prioritize space technology issues. Previous efforts in this area showed that quantitative methods inadequately modeled the space R&D effort. However, during the course of this research I found that traditional DSS implementation strategies may have serious shortcomings. I address this problem by advancing a new implementation strategy. I hope that this strategy may one day be used successfully in many organizations.

This report is limited in scope to implementing a small portion of the total space portfolio selection process. The specific methodology implemented in this report was developed by Capt. John Puffenbarger in a similar thesis effort. Included in this report are: A description of decision support systems, a review of previous implementation strategies, an explanation of the "pilot model" implementation strategy, and a pilot DSS demonstrating the major principles and processes of a decision support system.

I would like to thank my advisor, Lt Col Mark M. Mekar for his guidance and support throughout this thesis effort. He has invested his time and effort to insure that this research was a valuable learning experience for myself and my sponsors, the AF Space Technology Center at Kirtland AFB, New Mexico. I would also like to thank Lt Col Pete Soliz of the Space Technology Center, who has taken the time to make this a truly useful effort for myself and his organization.

Last, but by no means least, I would like to thank my wife Cecily for her love, patience and support throughout the course of the graduate program and especially this thesis effort.

CONTENTS

Preface	ii
List of Figures	vi
List of Tables	vii
Abstract	viii
I INTRODUCTION	1
Background	1
Problem Statement	4
Research Question	4
Objectives	4
Scope	5
II LITERATURE REVIEW	6
Introduction	6
DSS Definition	6
DSS Implementation	8
Evaluation of DSSs	12
AFSTC Decision Process	12
III GENERAL DSS IMPLEMENTATION STRATEGY	14
Current Strategies	14
Proposed Strategy	19
IV PILOT MODEL DEVELOPMENT CONSIDERATIONS	26
Implemented Decision Process	26
System Hardware Considerations	35
System Software Considerations	36
V IMPLEMENTATION DESCRIPTION	41
Program Description	41
Program Performance	56
Final Comments	57
VI ANALYSIS OF THE PILOT MODEL DSS IMPLEMENTATION STRATEGY	58
Pilot Model Implementation Goals	58
General Comments	62
VII CONCLUSIONS AND RECOMMENDATIONS	65

Bibliography	68
Appendix A: Verification of Computational Algorithms . . .	70
Appendix B: User's Manual	73
Appendix C: Pilot DSS Source Code	84
Vita	162

List of Figures

Figure

1	MSSTP Hierarchical Flow	2
2	DSS System Architecture (Sprague and Carlson)	10
3	DSS System Architecture (Denise)	11
4	DSS System Architecture (Ginzberg and Stohr)	11
5	MSSTP Hierarchical Flow	27
6	Methodology for Determining the "best" Concept Option	28
7	Relationship of Technology Issues to Concept Option	34
8	STC DSS Functional Flow	44
9	Database Management Options Menu	47
10	Concept Options DBMS Options	48
11	User Database Review Options Menu	51
12	VIEW Menu, first Option of Review Section	52

List of Tables

Table

1	MSSTP Definitions	3
2	Implementations strategies, (advantages and Disadvantages)	17
3	AHP Scale of Relative Importance	30

Abstract

The Air Force Space Technology Center is responsible for defining goals, tasks and priorities for the military R&D effort in space. To accomplish this task, the Space Technology Center has developed the Military Space Systems Technology Plan (MSSTP).

Past research has shown that the research and development portfolio selection process does not lead to an easy quantitative analysis solution. The very nature of research and development issues leads to a more direct involvement in the process by the decision-maker. The concept of decision support systems (DSS) is tailored toward unquantitative, decision intensive problems.

The objective of this research was to begin implementation of a DSS for the Space Technology Center. Research indicated, however, that traditional implementation strategies may have serious drawbacks to their effectiveness. To overcome the perceived drawbacks to the traditional implementation approaches, a new implementation strategy was formed. This strategy is called the pilot approach.

This research develops the new strategy, begins its implementation and performs an analysis on the system after beginning the implementation process at the Space Technology Center.

I. Introduction

Background

The Air Force Space Technology Center (STC) is responsible for defining goals, tasks and priorities for the military research and development (R&D) effort in space. To accomplish this task, the STC developed the Military Space Systems Technology Plan (MSSTP) (4:1-1).

The MSSTP was originally intended as an attempt to optimize the investment of resources for space technology research (21:3-1). It catalogs space related technology information, lists possible space system threats, identifies mission requirements, translates mission requirements into performance parameters, identifies technology shortfalls between performance parameters and current capabilities and recommends R&D programs to meet the projected shortfall (4:1-1). The stated purpose of the MSSTP is to identify and advocate space technology needs so that the necessary technologies are available when they are needed to support military space system performance requirements (4:iii).

The MSSTP is not just a collection of facts, but a process in and of itself. The purpose of this process is, as previously stated, to prioritize technology issues. Figure 1 is an illustration of the MSSTP hierarchical flow and Table 1 is a list of definitions. The MSSTP process, as evidenced by Figure 1, flows logically from perceived missions in space to technology plans that intend to provide the necessary technology to meet the future need. When the MSSTP was first written in 1981, a resource allocation model called the Technology Resource Utility Management process (TRUMP) was designed to be used with the data in the MSSTP. TRUMP was a set of decision rules intended to

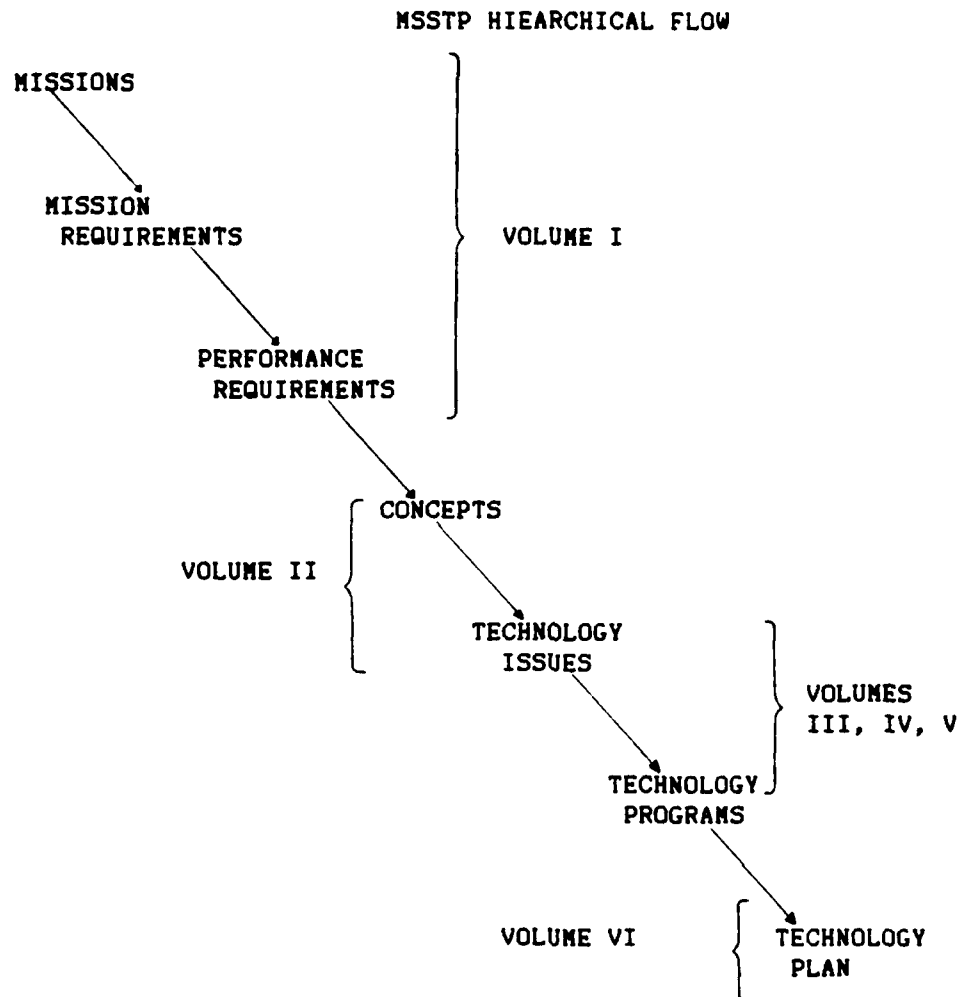


Figure 1. MSSTP Hierarchical Flow (4:iv)

provide an optimum strategy for resource allocation based upon the input priorities of the technology programs and their cost. TRUMP, however, was soon discarded as a viable model because it became obvious that it was attempting to develop a priority list by applying quantitative methods to a strategic planning problem, which is unquantitative in nature (21:3-4,3-5,3-27). In a Masters thesis for the Air Force Institute of Technology, Rensena and Chapman developed a decision support methodology for prioritizing space technology issues,

TABLE 1
MSSTP DEFINITIONS

- MISSION
 - A general military statement formulated to counter identified threats and meet needs.
- MISSION REQUIREMENT
 - A specific statement of what must be done to accomplish a mission.
 - Two or more mission requirements per mission.
- PERFORMANCE REQUIREMENT
 - Those quantified capabilities necessary for the accomplishment of military mission requirements. Specifically, the coverage, capacity, quality, timeliness, availability, and survivability of the space concept should be addressed.
- CONCEPT
 - A general outline of a space system with performance goals and technology requirements.
- TECHNOLOGY REQUIREMENT
 - Capability needed to meet desired concept performance.
- TECHNOLOGY ISSUE
 - A technology requirement beyond projected state of the art necessary for space system performance.
- TECHNOLOGY PROGRAM
 - A specific program which addresses a technology issue. Most technology issues have more than one technology program associated with them.

(4:v)

specifically for the MSSTP (21:1-1). This methodology recommended the use of a computer aided decision making tool as the means to effectively prioritize space technology issues (21:6-1). A computer-based decision making aid is commonly termed a Decision Support System (DSS).

Problem Statement

A decision-making aid would be helpful to the Space Technology Center (STC) to help prioritize and advocate the technologies presently listed in the MSSTP, and any future technology issues. An effective implementation strategy will be needed to create and maintain advocacy for a complete structuring of the STC decision process. The decision-making aid should be designed for use on a microcomputer system to insure that it can be used by resources currently present at STC. A pilot Decision Support System, designed for a microcomputer system, would prove beneficial to the appropriate decision makers at STC in making more effective decisions.

Research Question

How can a pilot decision support system be designed for the Space Technology Center to best demonstrate the capabilities of the DSS concept to provide a basis for advocacy of computer-aided decision support for the STC decision process?

Objectives

The major objective of this research is to develop an effective strategy for the implementation of a decision support system for the prioritization of space technology issues at the Space Technology Center. Specific subobjectives are:

- 1) Develop software that demonstrates the major characteristics of a decision support system.

- 2) Demonstrate the pilot model DSS to the STC to begin the DSS development cycle at STC.

Scope

This research is intended to provide an effective strategy for the development of a DSS at the Space Technology Center. It is also the intent of this research to begin the first step in the implementation process and develop a series of software capabilities that demonstrate the main characteristics of a DSS. This pilot model DSS will be developed to provide a basis for advocacy of the DSS concept for use within the STC organization. This research will address shortfalls in previous implementation strategies and introduce a new strategy that will overcome them. It is intended that this implementation could be the nucleus of a full support microcomputer implementation of a DSS for the prioritization of military space technology issues.

II. Literature Review

Introduction

In recent years, the term Decision Support System (DSS) has become a major topic of discussion. The definitions, characteristics, components and history of DSS are all hotly debated and vary a great deal in the literature. It is not the purpose of this literature review to discuss in detail the complete block of knowledge available concerning Decision Support Systems. In fact, many good literature reviews and summaries concerning DSS already exist (9;16). Instead, this review will focus on a basic, generalized overview of the DSS literature and only provide specific information on those points that are relevant to this research.

Decision Support System Definition

Since the early 1970's a number of authors have advanced different definitions of the DSS concept (1;3;9;10;14;17;18). For example, in 1971 Gorry and Scott Morton (10) identified DSS as systems that would support managerial decision-makers in unstructured or semi-structured decision processes. The key concepts advanced by Gorry and Scott Morton in the above definition were support and unstructured. The systems that they wrote about were aimed at extending the capabilities of the manager; not replacing him or his judgement. Thus the emphasis is on support. These systems were also aimed at helping managers solve unstructured problems, problems that could not be coded into an algorithm and solved automatically (9;9). A structured problem is one in which the decision being made is repetitive and routine "to the extent that a definite procedure has been worked out for handling them

so that they don't have to be treated de novo each time they occur (13:85)." In contrast, an unstructured problem has some distinctive attribute or attributes that makes it unique each time it arises, making it impossible to deal with automatically. An implicit assumption of most, if not all, early definitions of DSS was that they should be computer-based (9:10). This is evident from the fact that the structured and unstructured definitions closely follow the definitions of Simon's terms "programmed" and "unprogrammed", respectively, and Simon's admission of borrowing the terms from the computer industry (25:6).

An extension of Gorry and Scott Morton's DSS definition was provided by Little (17). He did not specifically use the term DSS, but defined "decision calculus" as a system of models to help a decision-maker solve decisions (9:10). Little also defined characteristics that such a system must have to be successful, as did Moore and Chang (18) and Bonczek, Holsapple and Winston (3), i.e., simple, robust, easy to control, adaptive, and complete (9:10). Finally, Keen (14) described DSS as an evolutionary process, with the user, the DSS, and the DSS designer interacting with each other (9:10). While all of these definitions provide pieces of the DSS picture, they also raise some basic questions. For example, Keen's (14) definition may lead us to ask "how the development process can be structured to assure that the feedback loops among user, builder, and system are in place and functioning (9:12)." Ginzberg and Stohr argue that although the more recent definitions (1;3;14;18) pose interesting problems, they change the emphasis of DSS research away from its central issue of supporting and improving decision making for managers (9:12).

The perceived move away from the real "issue" of DSS lead Ginzberg

and Stohr to propose the following definition for a Decision Support System:

A DSS is a computer-based information system used to support decision making activities in situations where it is not possible or not desirable to have an automated system perform the entire decision process (9:12).

It is the above definition that will be used to define a Decision Support System in this research paper.

A literature review from a different perspective may be found in a thesis by Koble (16) and a history of the DSS concept in Clark (5). Both show the DSS as a continuing process developed from electronic data processing (EDP) and management information systems (MIS). It is argued that DSS has developed as the role of computers in managerial organizations has developed. The work of Davis and Ohlson (7) also support this evolutionary view.

Decision Support System Implementation

While there are many opinions on the definition of DSS, there are as many or even more opinions on what features constitute a DSS. This literature review will focus on the characteristics of a DSS that may be useful for the type of implementation planned.

Some of the most important characteristics of a DSS that are seen constantly in the literature are:

1. Man-machine interface, where the decision-maker retains control over the decision process.
2. Utilization of the appropriate mathematical and statistical models.
3. Query capabilities to obtain information by request.
4. Comprehensive database with database management.
5. Easy to use approach and a completely user friendly interface between the decision-maker, the

database, and the models.

6. System is adaptive over time, i.e. flexible.
(2;7;9;13;27;29)

The above characteristics drive system architectures that are different, yet very much the same. For example, Figure 2 is a system architecture by Sprague and Carlson. It is composed of a database management system (DBMS), a model-base management system (MBMS), and a dialog management system (DGMS). The DGMS is another name for the user interface. Figure 3 is from an article by Denise. His system architecture is structured slightly differently than Sprague and Carlsons. The report block represents the user interface. The modeling system is represented by two blocks, statistics and modeling. The database system is represented as the database block. Finally, Figure 4 is from Ginzberg and Stohr. It adds a few extra sections to the system configuration. The extra blocks, however, are not unique to their system representation. They are implicit in the definition of the user interface for the first two system architectures. It is interesting that the models all include databases, user interfaces and modeling capabilities. They only differ in how the different components are addressed by the DSS, which is by no means a minor point. They do, however, give a strong indication of the necessary components, i.e. a database, a modelbase, and a user interface, that should be included in a DSS, and what their capabilities should be. An interested reader can find further DSS model representations in Thierauf (29), Alter (2), and Bonczek et al. (3). After researching what a DSS is, what characteristics it should have, and its most likely form, it was necessary to look into its applicability to personal (micro) computers. It has been recognized that (potentially, at least) the microcomputer represents the single most important advance

THE DSS

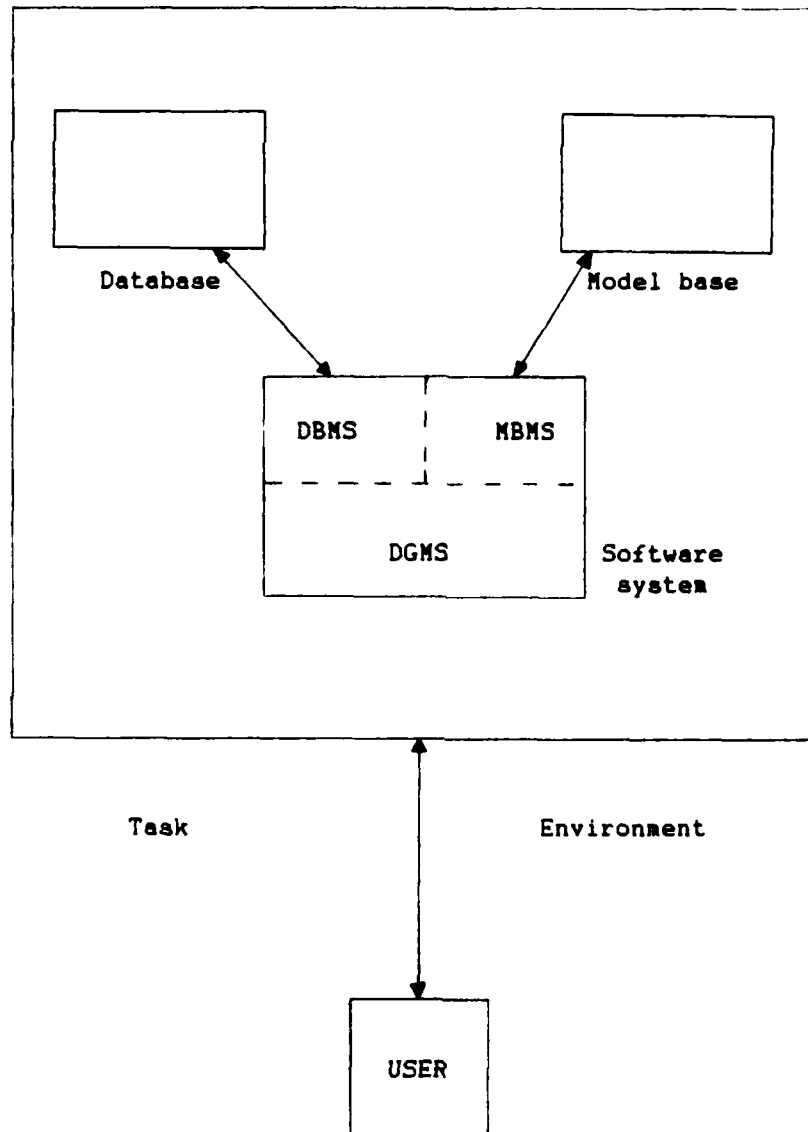


Figure 2. DSS system architecture - Sprague and Carlson (27:29)

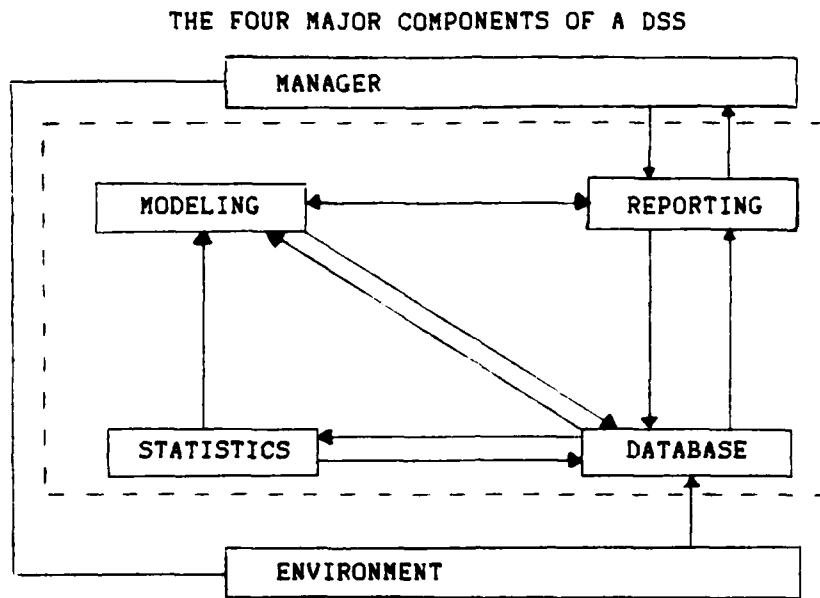


Figure 3. DSS system architecture - Denise (8:208)

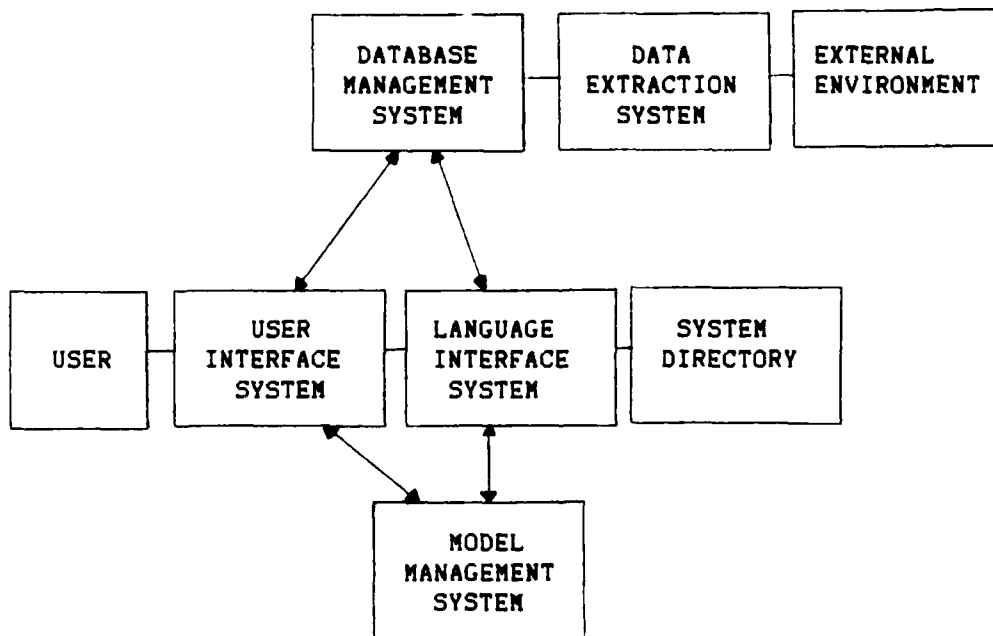


Figure 4. DSS system architecture - Ginzberg and Stohr (9:17)

in the DSS area. As the speed and capability of the microcomputer increases, so does their utility to the decision-maker. For the first time the decision-maker can bring computing power into his office under his personal control (8:210;15:34). The utility of using microcomputers to solve Operations Research problems was the topic of a thesis by Greg White. In his research he found a number of articles that supported this view (30:8). Nobles (19) also came to the same conclusion in a similar effort.

It is clear at this point that the literature supports microcomputer based decision-aids in the form of decision support systems. There is also a large body of knowledge agreeing on what these systems should do.

Evaluation of Decision Support Systems

One area of operational interest that has not received much attention is that of DSS evaluation. Bertram Spector, an analyst with a firm practicing DSS techniques for the corporate world, feels that this is one of the most pressing areas for DSS research in the coming years (26:1). Sprague and Carlson agree, stating that few, if any, evaluations of DSS systems have been reported (27:157). They list several possible ways that a DSS could be evaluated, such as cost-benefit analysis, value analysis, cognitive testing and attitude surveys (27:162). Both Sprague and Carlson and Keen and Scott Morton agree, however, that the only practical measure of DSS success is if it is extensively used (13:216;27:158).

AFSTC Decision Process

An extensive review of the AFSTC decision process was accomplished

by Rensema and Chapman (21). They also completely reviewed the previously mentioned methods for prioritizing space technology issues. In their work, they found that the Analytic Hierarchy Process (AHP), a method of ordering alternatives, was preferable to other methods of prioritization for the space technology problem (21:2-27,2-28). In developing a methodology to identify the "best" point design among concept alternatives (see Chapter 1), Puffenbarger also selected AHP as the decision process of choice (20:27).

III. General DSS Implementation Strategy

In general, a Decision Support System (DSS) development effort requires a number of people to be involved at many different levels of the effort (27:64). Also, DSS implementations are generally considered to be iterative processes, where system prototypes are developed, evaluated, and redeveloped due to end-user inputs (11:80). However, as stated earlier, it is not the intention of this research to "prototype" a DSS. This research develops a "pilot" model; a model that demonstrates the basic principles of a working DSS. This new implementation strategy and how it fits into the architecture of current iterative development schemes are the subjects of this section. First, however, several current strategies for DSS implementation will be profiled.

Current Strategies

Most of the many different implementation schemes advanced in the literature could be placed (to some degree or another) into the following three categories: the "quick hit", the staged development approach, and the complete DSS [as advanced by Sprague and Carlson] (27). All of these strategies stress the importance of the evolutionary process that is characteristic of a DSS. In fact, recent literature has gone "as far as saying that an evolutionary design process is a prerequisite for calling a system a DSS (9:24)."

Quick-Hit. The first strategy advanced is the quick-hit. This strategy advocates the development of a working DSS in an area where there is an easily recognizable payoff. This would involve using the most appropriate tools, and reaping the benefits as quickly as possible. This strategy has some obvious strengths. First, the quick hit method

provides immediate benefits. The organization is able to use a DSS for a specific task immediately, producing a fast, measurable payoff.

Secondly, this strategy has an associated lower risk. The development of the DSS is "quick", reducing the chance that the decision process changes during development. The chance of technological obsolescence once the system is complete is also minimized.

Third, the quick-hit strategy has easier development procedures. The quick-hit method requires less planning than other methods because it is not meant to be expanded. When a new system is required, it would be built from scratch.

Finally, the quick-hit strategy can take advantage of the availability of the latest technology. The DSS can be constructed with the very latest hardware and software available.

After the first DSS is in place and functioning at its intended purpose, the organization should begin the second iteration of the implementation process. The changing requirements of the completed system, as well as new areas in the organizations decision process, would be identified and included in the design of the next DSS. The second (and subsequent) iteration DSS would use the latest tools and technology available to implement a completely new system.

With advantages, however, come disadvantages. The fact that there is no carryover from the first DSS to subsequent DSS implementations is a disadvantage as well as an advantage. The software developed for the first DSS may not be applicable to the next DSS. Also, a DSS developed using the most expedient means may not be as flexible as a DSS built using another strategy. Therefore, a quick-hit DSS may require more maintenance to keep it current (27:60-63). Some of these disadvantages

are overcome by the second implementation strategy, the staged development approach.

Staged Development Approach. The staged development approach tries to combine the quick-hit method with more planning and forethought. This strategy involves developing the first DSS with its application toward the second and subsequent systems kept firmly in mind (27:60). This strategy hopefully leads to the development of a DSS Generator. A DSS Generator is defined as "a package of related hardware and software which provides a set of capabilities to build specific [working] DSS quickly and easily (27:11)." The main advantage of this approach is that it gives an organization an initial DSS capability, but does not sacrifice expandability. The initial DSS is a step toward a DSS generator that can be rapidly applied to many organizational decision scenarios. Also, this approach would allow the integration of new technology without losing the previous capabilities as older systems are scrapped. At the same time, the organization's risks of technological obsolescence and changing decision requirements are minimized by the fairly quick development of the first working DSS. The disadvantages, however, are that the organization must place some initial resources on the development strategy and experience some delay in reaping the benefits (27:61-62). The final implementation strategy is the most general of all and is called the complete DSS by its proponents.

Complete DSS. The complete DSS implementation strategy proposes that major planning and development should be accomplished before the first working DSS is built. This strategy would entail the development of a complete DSS generator, as defined earlier. Again, there are advantages and disadvantages to this implementation scheme. This

TABLE 2

QUICK HIT

ADVANTAGES

- Fast payoff
- Low risk
- Easier to apply technology and development procedures
- Latest technology always available

DISADVANTAGES

- One shot-no likely carry over to next DSS
- Specific DSS may require more effort to change

STAGED DEVELOPMENT

ADVANTAGES

- Leads to development of DSS Generator
- Gives early success and visibility
- Allows integration of new and old DSS
- Ability to assimilate evolving technology

DISADVANTAGES

- Requires additional cost up front
- Delays initial success somewhat

COMPLETE DSS

ADVANTAGES

- Likely to be best integrated and have best architecture
- Will reach full strength soonest

DISADVANTAGES

- Long development time before first benefits are realized
- High risk of technological obsolescence
- High risk of unknown problems

Implementation Strategies: Advantages and Disadvantages
(27:62)

approach has a greater probability of becoming the most efficient generator. It will probably have a better integrated set of tools to work with and it will probably have a better system architecture. The complete DSS approach will probably also lead the building organization to "full-up" status first. That is, this approach will give the organization its most complete, and best organized, DSS generator and specific decision support systems quickest, in the long run. The disadvantages, however, are considerable. A complete DSS development effort will probably take a long time, and is considered a multiyear program. Because of this lengthy development time, the complete DSS is susceptible to becoming technologically obsolete before the first working DSS is constructed from the generator (21:63). Table 2 summarizes the major advantages and disadvantages of the implementation strategies as outlined in the paragraphs above.

Sprague and Carlson, after reviewing the three implementation strategies listed above, conclude that the staged development approach is the best as it balances the other two schemes (27:63). In fact, most DSS researchers seem to agree. Thierauf's (29) implementation scheme follows closely in most major details with the staged development approach. It recommends the development of a first complete system with appropriate planning, and then the integration of more capabilities later (29:131-147). Keen and Scott Morton also follow the staged development approach, although they term the phrase quite differently: as the predesign cycle, design stage and postdesign process (13:167-185). Their approach also advocates the development of a working model and increasing capabilities through the iterations. Neither Thierauf or Keen and Scott Morton specifically mention the concept of DSS gener-

ators, but the idea is implied in the continually expanding system. There is evidence, however, that the staged development implementation strategy, and all prototyping strategies in general, may have even more serious shortcomings than previously mentioned. These shortcomings are dangerous because they are manifested in characteristic strengths of DSS implementation strategies. In a recent study, two researchers (Henderson and Ingraham) found that prototyping is a "highly convergent design process (11:86)." This type of process, they feel, would lead to a high degree of user satisfaction and therefore, use, which is one of the previously discussed objectives or characteristics of a DSS (see chapter 2). They also found that the availability of a specific DSS ready for use fairly quickly is useful in supporting the user during the lengthy development time for the system generator (DSS generators are again not expressly mentioned, but are implied). The problem lies in Henderson's and Ingraham's findings that this process may lead to information requirements and needs being missed, with the propagation of "a status-quo which is sub-optimal (11:86)." Furthermore, by focusing the design of the initial DSS on a single user, the entire system may reflect the bias of that one individual throughout the life of the system. Finally, they found that the prototyping approach (the staged development approach) is particularly effective in defining system integration needs. The single user focus, however, may not be effective in implementing those requirements into the integrated DSS (11:86).

Proposed Strategy

In this research effort, a new implementation strategy will be advocated and initiated for the Space Technology Center. This implementation strategy will be similar to the staged development approach with

leanings toward both the complete DSS approach and the quick-hit approach. The new strategy will be called the "pilot" approach.

In general, the pilot approach to DSS implementation will be a parallel effort. Once an organization has decided that a DSS has potential applications for some of their decision processes, a DSS working group (much like that for staged development) would be formed. As in the staged development approach, this group would consist of users, Operations Research analysts, and programmers. It should be noted that whether or not this development will be accomplished "in-house" or by contract has no bearing on the composition of the working group. If the DSS is to be built in house, the interested parties will be assembled from different divisions. If the work is to be accomplished by contract, the users will come from the buying organization with perhaps representation from other departments, such as the programming department, if the system will be maintained in-house after purchase.

After the working group is formed, the major difference between the prototype approach and the pilot approach is evident. At this stage, the prototype approach would begin working on a preliminary study before developing the first specific DSS. Under the pilot approach, the group would begin a preliminary study, but would also immediately develop a "pilot" DSS. The pilot model DSS would be an actual DSS capable of functioning and providing an initial, although quite limited, capability to the end user. It is important to note that the pilot model need not be a complete working model. The intent of the pilot model is to serve as a basis for DSS understanding. The pilot model would be built with no intention of follow on DSSs, as is planned in the staged development approach.

The pilot model should contain as many features as possible, but remain simple so that it is put together quickly. At the same time, or perhaps after the pilot model is completed, a complete DSS specification and implementation strategy would be developed.

As previously stated, the main objective of the pilot model is to provide a basis for understanding the nature and characteristics of a decision support system within the organization. To support this, the major characteristics that the pilot model should display are:

- 1) User friendliness. The major emphasis in designing and implementing the pilot model should be the man-machine interface. If the program is not easy to use, it won't be used.
- 2) Appropriate operations research techniques, mathematical and statistical models. The pilot program should contain enough capability to provide an accurate picture of the potential capabilities of the eventual "full-up" system for the end-users.
- 3) The usefulness of databases and database management. The DSS software for the microcomputer should be able to provide an accurate representation of the information needs of the organizational decision process, and how the information will be managed and presented.

The characteristics listed above are several of the most important for any DSS. They are not, however, all of the characteristics commonly listed in the literature (see Chapter 2). Not all characteristics of a working DSS are necessary in a pilot model. For example, the flexibility of a DSS is usually a major factor in its development. A DSS should be flexible internally and externally. Internal flexibility means that the software itself is flexible for the user. It would allow the user a wide range of options and be able to perform a large number of tasks. The DSS would perhaps allow the user to link functions and create new tasks. External flexibility concerns the programmers of the

DSS. To be externally flexible the DSS should be easy to change and iterate as a system. For these reasons, there is no need to emphasize flexibility in the demonstration model. The program should be flexible enough to display the concept of internal flexibility to the user, but should also remain simple. To remain simple, the program should not contain external flexibility features that would increase the development effort of the programmers and designers of the pilot model.

The pilot approach, as stated, would remove most of the disadvantages cataloged for the other implementation strategies. In fact the objectives of the pilot model are to overcome the limitations of "prototype" strategies, such as the staged development approach. Simply stated these objectives are: 1) Educating users about DSS's and their capabilities so that the users may better define their requirements (type and format of data, type of analysis, possible functions, type of user interface, displays, etc..) for the complete system; 2) Generate acceptance and possibly advocacy, which are keys for DSS success, and generate further interest in other users, to eliminate single user bias; 3) Give users, programmers and designers, the main components of the DSS working group, an appreciation of the other group members point of view and to stimulate knowledgeable discussion between them while the DSS project is in its infancy, thus avoiding costly communication errors; and 4) As most decision support systems are designed around existing databases, a pilot model would give the designers/programmers an understanding of the current database. This understanding would include issues such as data content and structure. For example, if information is stored as characters, but must have arithmetic operations performed on it to present it in a format palatable to the decision-maker, it must

be converted to a numeric format.

In summary, the objective of the pilot system would be to inform the organization of the many possibilities inherent in DSS, and to channel the interest into the development of a DSS generator that could benefit the entire organization, not some small part of it. Secondly, while educating all people that may be involved in the development process, the pilot model would point out at an early stage development problems that need to be addressed.

On one hand, it has been shown that the prototype strategy for DSS development has some serious shortcomings. On the other, it has some major advantages also. The pilot approach, as advanced in this research, is designed to eliminate those disadvantages. No technique, however, is without its own disadvantages. Some may consider it a waste of resources and effort to develop a system that may not provide any real capability, with the intention of discarding it at a later date. These critics would be at least partially correct. The issue is dependent on the amount of return for the investment. By creating interest in the organization and demonstrating system capabilities, a more cost effective system is achievable. If, for example, the staged development process is followed for a number of years, and the user that has been the major focal point of the DSS and its many iterations leaves the organization, the previous effort may contain too much personal bias to be palatable to his or her replacement. In contrast, the pilot approach attempts to increase the number of interested users in the organization. Also, the purpose of the demonstration model is to promote the education of the user. If a new user arrives after the pilot model has been constructed, and the implementation process is nearing completion of the

first working model, the pilot model can be used by the designer to identify any new requirements generated by the new user. This point is extremely critical with respect to an organization such as the military, which experiences frequent personnel changes in its upper and middle management.

It should be noted that the pilot model implementation strategy may not be effective when applied to small organizations. This strategy would not be cost effective for a small organization that does not have a wide range of decision processes or possible users of the system. The decision to use a pilot model implementation approach is based upon the possible gain in knowledge of the user versus the potential costs. A small organization may not be able to justify the use of resources to build a pilot model.

The pilot model strategy may be particularly effective when applied to large organizations. Large organizations may have a multitude of decision processes. By generating advocacy within the organization the pilot model approach may lead to an integrated DSS for the entire organization. An integrated DSS may save the organization significant costs over the life of a DSS implementation effort.

As a final point, a pilot model implementation approach may indicate that further DSS development is inappropriate. A particular problem may be insurmountable with current technology, or the decision-maker may decide that a DSS is not needed by the organization. The pilot model approach may be a particularly valuable strategy when the organization is not sure that it needs a DSS. Building the pilot model could act as a catalyst that would bring the organization to a swift decision, either for or against further development. In either case,

the pilot model would have served its purpose. An organization would have made its decision with the least cost invested in the development effort.

Because of the above advantages, the pilot model implementation strategy is a valuable approach to DSS development in a military environment.

IV Pilot Model Development Considerations

Implementing a pilot model decision support system for space technology tradeoffs required a careful analysis of the decision process at the Space Technology Center. As in any DSS design and development phase, the pilot approach included, user involvement is essential. The portion of the MSSTP process selected for implementation had to be the area most likely to receive the highest degree of user support. This section first describes the area selected for implementation and the reason for doing so. System hardware and software considerations are also discussed.

Implemented Decision Process

The decision process selected for implementation was an extension of the current MSSTP process developed by Puffenbarger in his thesis "A Methodology for Assessing Technology Trade-offs of Space-Based Radar Concepts." In short, Puffenbarger asserts that another step is needed in the current decision process. He contends that concepts, as defined in Chapter I, should be broken into concept options (which are point designs that satisfy the general requirements of a concept). In choosing the best concept option, using a predefined set of criteria, the technology issues associated with that concept option would have more weight in the overall prioritization scheme (20). This methodology was chosen for implementation for two main reasons. First, STC had expressed great interest in a process that helped choose the best concept option or point design of a given concept. This interest could work to the advantage of a pilot model type implementation strategy by generating interest in decision support systems, once a system is developed.

Secondly, as Puffenbarger's research was conducted concurrently with this effort, a high degree of involvement by a "user" could be achieved in the formation of the pilot model. As advances were made in the methodology, changes could be made in the pilot model. A further explanation of the methodology is given below.

The new methodology expands the concept to technology issue portion of the MSSTP's current process by adding what Puffenbarger terms "concept options" (20:3). Puffenbarger argues that different system

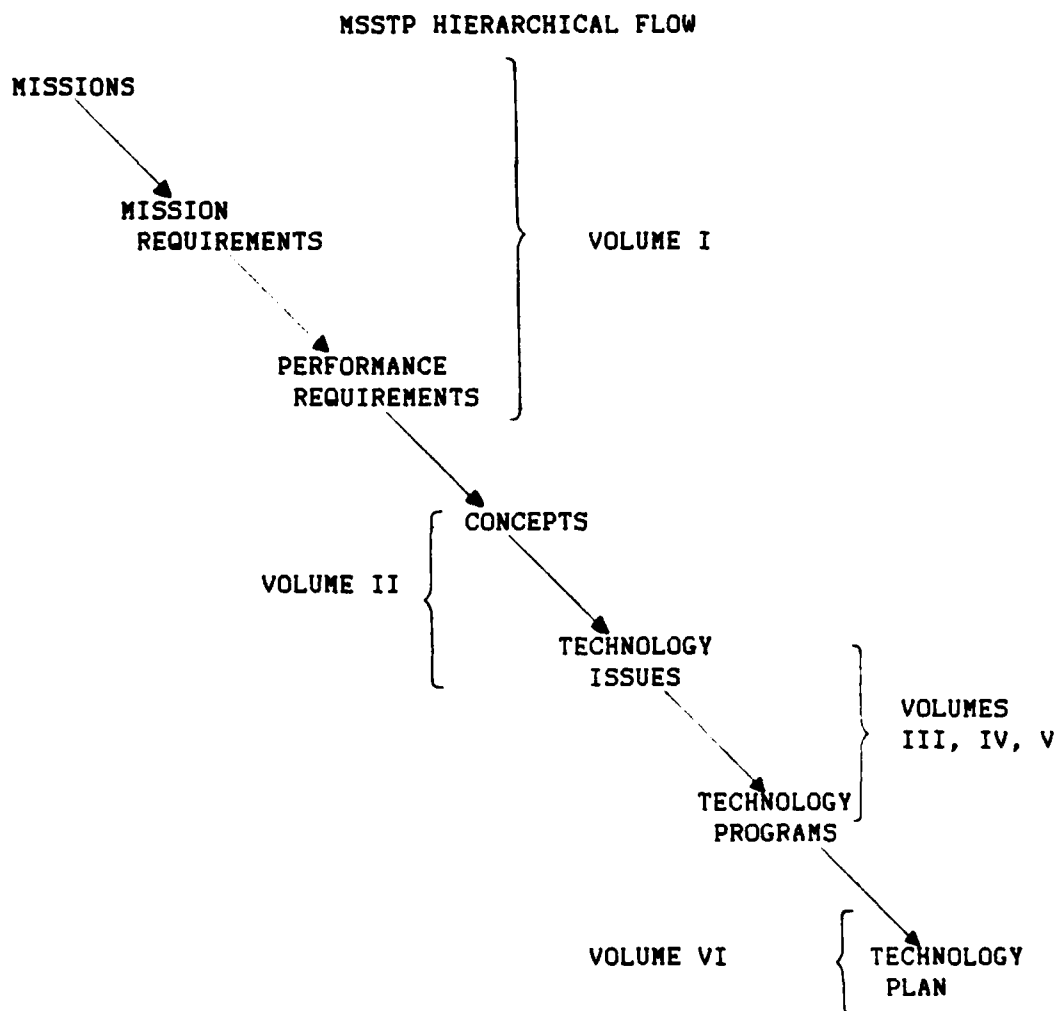


Figure 5. MSSTP Hierarchical Flow (4:iv)

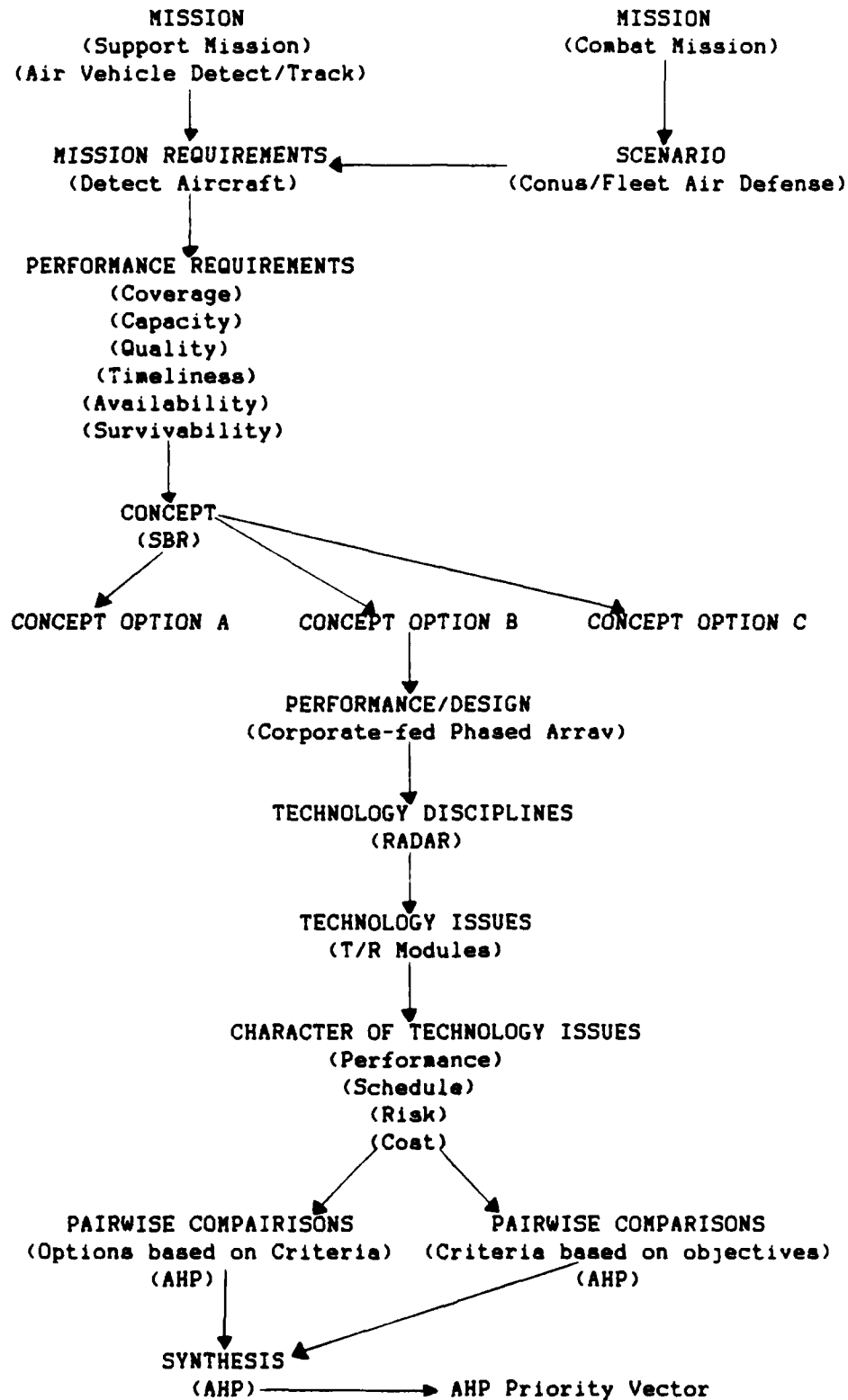


Figure 6. Methodology for Determining the "Best" Concept Option (20:96-99)

designs influence the rank ordering of technology issues. If the best system point design can be found, then the technology issues associated with that design (concept option) should be ranked the highest in priority for the given concept. Figure 5 is the representation of the MSSTP planning or prioritization process from the MSSTP itself, and is a copy of Figure 1 from Chapter I. Figure 6 is the methodology for determining the "best" concept option, as developed by Puffenbarger. The two processes are similar until the concepts (space system concepts) are determined. At that point, Puffenbarger breaks a concept into concept options. Then, Puffenbarger defines four criteria with which the different concept options can be measured against: performance, schedule, risk and cost. The performance of a concept option is the estimated performance of the concept option in six areas as defined in the MSSTP, i.e., survivability, coverage, capacity, quality, timeliness and availability. The schedule for a concept option would include the earliest completion date, earliest production date and initial operating capability (IOC). The risk for a concept option could include factors such as the number of "risky" technology issues associated with a concept option, the potential variability of the schedule for solving the technology issues associated with the concept option, and the potential failure of one or more technology issues in meeting the required performance levels. Once these criteria were defined and the concept options were identified, Puffenbarger chose the Analytic Hierarchy Process as the methodology best suited to providing a structured decision process to this problem (20:55-80).

The Analytic Hierarchy Process (AHP)

is a systematic procedure for representing the elements of any problem. It organizes the basic rationality by

breaking down a problem into its smaller constituent parts and then calls for only simple pairwise comparison judgements to develop priorities in each hierarchy (22:140).

Translated for simplicity, AHP, as applied to the methodology outlined above, breaks the problem of finding the best concept option into two parts. First, the relative weight or priority of the criteria must be found. Then, the options must be compared against each other with respect to the criteria, to find the relative weight or priorities of the concepts with respect to each criteria. The relative priority of each element of a given level of the hierarchy is found by first presenting a series of pairwise comparisons of the elements. For

TABLE 3

SCALE OF RELATIVE IMPORTANCE

Intensity of Relative Importance	Definition	Explanation
1	Equal Importance	Two activities contribute equally to the objective.
3	Moderate Importance of one over another	Experience and judgement slightly favor one activity over another.
5	Essential or Strong	Experience and judgement strongly favor one activity over another.
7	Very Strong	An activity dominates the other, and demonstrates it in practice.
9	Absolute Importance	The evidence favoring one activity over another is overwhelming.
2,4,6,8	Intermediate values	When compromise is needed. (22:145)

example, the first level of the hierarchy of Puffenbargers methodology is the attributes criteria, performance, schedule, risk and cost.

The comparisons are based upon a scale that is represented in Table 3. This scale defines the relative importance of one object to another, and is the value that is used for priority calculations. The comparisons build a square, reciprocal matrix at each level of the hierarchy. This matrix is solved for its characteristic eigenvector. This eigenvector is then normalized by dividing its components by the sum of all elements of the eigenvector. The normalized components of the eigenvector are then the priority vector for the elements of that level of the hierarchy (22:141).

The synthesis of priorities is performed after all of the priority vectors have been calculated. The overall priority of the hierarchy is calculated

by multiplying local priorities by the priority of their corresponding criterion in the level above and adding them for each element in a level according to the criteria it affects. This gives the composite or global priority of that element (22:141).

For the methodology of Puffenbarger, this means that the priority or weights of the four attributes, performance, schedule, risk and cost is calculated. Then, the priority of the concept options is calculated based upon each of the attributes, resulting in four separate priority vectors. The weight of performance is multiplied to the concept option priorities based upon the performance measure. The weight of the remaining three attributes is multiplied to the appropriate vector. The final priority of each concept option is the sum of the weight of each concept option after it has been multiplied by the weight of the attributes.

An advantage of this technique is that a measure of the consistency of the decision-makers judgements is available. The consistency ratio (CR) measures the departure of the maximum eigenvalue from the number of elements at a given level of the hierarchy. The consistency ratio is found by first obtaining the consistency index. The consistency index is calculated by subtracting the number of elements (n) from the maximum eigenvector and dividing by ($n - 1$). The consistency index is then divided by a random consistency calculated n elements, yielding the CR. This measure should be below 10% to indicate consistent judgements by the decision-maker (22:142-143).

It should be noted that the above is a very cursory explanation of the Analytic Hierarchy Process. It is actually a rigorous and complex process designed to set structure to complicated and unstructured problems. The purpose of this review was to set forth a minor explanation of how AHP works so that its implementation in this research effort could be better understood. Explanations of the process and a study of its relative merits compared to other approaches can be found in many other works. Some deal mainly with the process itself (22;23), while others, such as theses by Puffenbarger (20) and Rensema and Chapman (21), deal with AHP in the specific setting of prioritizing technology issues. These references can be reviewed for a greater degree of understanding of the process itself. Finally, in developing a pilot model for the above methodology, several restrictive assumptions were made concerning the attributes and the concept options that would be presented.

First, for this implementation, it was assumed that the performance measures were equal in value. That is, timeliness is equally as

important as survivability. Although this may not be the case, it simplified the model by removing a level of the hierarchy. Each option was compared on the basis of performance as a whole, not on each separate performance measure.

A second assumption was that the schedule, risk and cost attributes were measured in terms of the unique technology issues of a given concept option. This means each concept option could be expressed as a group of technology issues. The schedule, risk and cost of a concept option is then the composite schedule, risk and cost of the technology issues.

For example, the three concept options for a given concept could each have ten technology issues in common. The first and second options could have five unique technology issues, and the third option have ten unique technology issues. The third option in this example (see Figure 7) has two issues in common with the first option, and two with the second. However, the respective technology issues are not contained in all three, so they are counted as unique. The decision-maker has to decide how the risk values of a concept option's unique technology issues compare to the risk values of another concept option's unique technology issues. The same argument applies to the schedule and cost attributes.

A third assumption concerning the implementation of the above methodology was that data existed in a format that could be used by the pilot model. Because the computerized version of the MSSTP database was not completed at the time of this research, an interface between the existing database and the DSS database could not be considered. It was assumed that the information, as displayed by the pilot model implemen-

CONCEPT

Technology Issues(TI)	OPTION A	OPTION B	OPTION C
1)	TI 1	TI A	TI 1
2)	TI 2	TI B	TI A
3)	TI 3	TI C	TI 2
4)	TI 4	TI D	TI B
5)	TI 5	TI E	TI U
6)			TI V
7)			TI W
8)			TI X
9)			TI Y
10)			TI Z

Risk of Option A is a function of the risk of technology issues 1 through 5. It can be expressed as a mean, and/or by the number of TI's with risk values in a certain category (very low to very high).

Figure 7. Relationship of Technology Issues to Concept Options.

tation, could be obtained from the finished MSSTP database if it were available. This restriction had some advantage, however. It stimulated thought into the structure and content of data displays and storage for the pilot model DSS. This was one of the objectives of the pilot model implementation strategy as discussed in Chapter III.

Finally, it was assumed that the target audience for the demonstration model would be familiar with the MSSTP and its associated decision process. This assumption reduced the information display and storage requirements of the DSS. This assumption did not stop the pilot model from meeting its major objective, educating decision-makers in the utility of decision support systems. The potential users and purchaser of the system (if they are not the same person) are located at STC, and

familiar with the MSSTP. An assumption such as this can be made if it does not significantly affect the performance of the pilot DSS.

The pilot model DSS is attempting to educate decision-makers and therefore must contain and use information familiar to its target audience. Concurrently, time, cost and effort savings in the pilot implementation effort are of great concern. These savings, however, can not be made at the expense of some capability that the pilot model must have to adequately inform the user. The bottom line is that a delicate balance must be maintained between too much and too little effort in the pilot model implementation strategy.

System Hardware Considerations

The objective of this research was to implement a decision support system on a microcomputer. There are however, literally hundreds of systems that can be classified as microcomputer systems (30:29). For this reason, the implemented DSS should be as "machine independent" as possible. This demonstration model was intended for the Space Technology Center, however, so the system hardware configuration should match the capabilities currently available at STC. As White discovered in his research: "Software which is developed on a system with limited availability to the target users will not be used extensively (30:29)." Thus, the hardware for the DSS implementation should be as close to current STC capabilities as possible, while the system itself is designed to be as independent and system flexible as possible.

The microcomputer selected for system implementation was the Zenith Z-100. The Z-100 was selected for several reasons. First, the Air Force has purchased hundreds of these machines in recent years, making them available for use at almost any Air Force location. This has

several advantages, such as the fact that a pilot model DSS designed and implemented on a Z-100 can be demonstrated at a multitude of Air Force units. A demonstration model of this nature is extremely important for organizations where the decision process is carried out at widespread geographic locations. For example, STC does not set the priorities of the concepts contained in the MSSTP. If Space Command were the designated command that performed this function, the pilot DSS could be demonstrated to them. The demonstration of the pilot DSS would hopefully generate advocacy for a DSS to help Space Command set the priorities of space system concepts.

Secondly, the Z-100 computer system supports a "basic" system configuration as defined by White (30:36-37). It is equipped with a video display screen, two disk drives, and over 64K of random access memory (RAM). It should be noted that the amount of RAM that the computer actually supports depends on the organization. In its basic configuration the Z-100 supports 256K of RAM and is easily expandable to higher levels of RAM. For DSS transportability, however, this implementation would take no more than 64K of RAM to run, meaning that the DSS could run on any system fitting White's definition of a basic system.

As a final note, it was assumed that the user of the demonstration model would have access to a printer. To gain the full appreciation of the DSS concept, hardcopy feedback was considered a necessity.

System Software Considerations

As transportability of the DSS was a major consideration in its design, the type of software chosen for implementation had to be commercially available and widespread in its use. It also had to be capable of modification to meet the demonstration objectives of the pilot model

(user friendliness, database management, and modelbase management - see Chapter 3). Although there are currently a multitude of high level programming languages available (BASIC, FORTRAN, COBOL, APL, ADA, Pascal and C to name some of the most popular), only a select few meet the requirements of being portable from microcomputer to microcomputer and in being machine independent in their implementation. The languages that were selected for consideration due to the above reasons were BASIC, FORTRAN, and Pascal. Other options were available, however. One of the primary objectives of the demonstration model DSS is to show database management principles and to educate users, programmers and DSS designers in the data requirements of the decision process. Several good database management packages exist for microcomputers, which also support their own programming languages. Of these, two were selected for consideration, dBase II by Ashton-Tate, and Savvy by Excalibur Industries. DBase II was considered because of its popularity with industry, and the fact that it can be supported by any microcomputer fitting Whites basic system configuration. Savvy was considered because it is the software that the MSSTP electronic database is being implemented in.

Of the programming languages selected for review, Savvy was the least favorable. It requires a minimum hardware configuration of an IBM PC, IBM PC jr., IBM XT, or IBM AT or a truly IBM compatible (100 %) system. Furthermore, the system requires at least 128K of internal memory (24:5).

BASIC was not selected because it does not have any type of advanced data storage capabilities. Also, it is machine specific in its file handling characteristics, making it less portable from machine

to machine. Finally, in most implementations, BASIC is an interpretive language. That is, the program code is executed line by line by a master program. Interpreted programs are considerably slower in execution speed than compiled programs, which is a further drawback (30:33).

FORTRAN was not selected because it lacks good data handling characteristics and is not a structured programming language. FORTRAN is much like BASIC in that it only supports limited data structures (30:34). It does provide extensive formatting capabilities for system output and has gained widespread use. The implementations, however, are many and the quality of some is than others, thus reducing program portability.

Finally, dBase II was not selected for several reasons. First, the programming language supported by dBase is an interpretive language, thus its execution is slower, especially when it must process many loops in the program code (likewise for BASIC) (19:7-4). Secondly, dBase II does not support arrays in its data structures (19:7-5). The calculations involving the Analytic Hierarchy Process involve the processing of arrays as matrix operations. To use dBase II, extensive extra file capabilities would have to be implemented. Finally, dBase II does not support many mathematical functions. Any mathematical functions needed by the program would have to be written in dBase II programming language.

Although each language has its advantages, it was for their disadvantages that they were not selected. It was determined that Pascal maximized the advantages of a programming language for microcomputer systems while minimizing its disadvantages. The Pascal version chosen for this implementation was Turbo Pascal by Borland International, and

it was chosen for the following reasons.

First, Pascal, in general, has excellent data handling characteristics. It not only supports the data types that the other high level languages do (real, integer and character values), but it allows the programmer the option of defining his own data types. These data types may be nested within each other to achieve maximum effect. For example, a record, a data type in Pascal, may contain a number of other data types. It may have other records, which have arrays, whose elements are other records, etc (6). The flexibility of data types was thus a major factor in the selection of Pascal.

Secondly, Turbo Pascal is the recognized de facto standard Pascal implementation. Borland International claims that over 300,000 copies of the program have been sold, which would make it the most widely used Pascal implementation (12:25). Also, Turbo Pascal is a general Pascal implementation. That is, a program that is written in Turbo Pascal will run on any other computer that supports Turbo, as long as none of the machine specific functions or options are used. The program code must be re-compiled for each computer the program will run on, however. This makes the language extremely flexible and portable from microcomputer to microcomputer. Finally, Pascal is a structured programming language. It is written in a series of program blocks called procedures that are the instructions of the program. The structured programming approach of Pascal encourages good programming techniques that make program maintenance and modification easier for someone unfamiliar with the program.

In summary, this demonstration model DSS implements a new decision process as developed by Puffenbarger. This process incorporates the Analytic Hierarchy Process as its central prioritization technique to

find the best space system configuration among a series of systems that represent a space system concept. The DSS was specifically configured for a Zenith Z-100 microcomputer system currently in use at STC. The DSS, however, will be capable of configuration for any microcomputer system that supports the Turbo Pascal compiler. The DSS was written in Turbo Pascal, the high level language that best helped fulfill the stated objectives of the pilot model.

V. Implementation Description

In Chapter 3, the goals and objectives of the pilot model implementation strategy were discussed. In Chapter 4, the decision process that was implemented, and hardware and software considerations were discussed. The purpose of this chapter is to synthesize the information of the previous two chapters by describing the program implementation and how it meets the objectives of the pilot model implementation scheme. This chapter begins with a general discussion of the program capabilities, and then describes in detail how the objectives of this research effort are achieved.

Program Description

The Space Technology Center's Decision Support System (STC DSS) was implemented using the hardware and software as described in Chapter 3. The overall purpose of the program (as implemented) is to allow a user to prioritize a list of concept options within a given satellite concept using the Analytic Hierarchy Process. The pilot STC DSS is subject to the following limitations:

- A total of 10 concepts
- 5 concept options per concept
- 15 technology issues per concept option.

The pilot model also is limited to the 4 criteria (attributes) that were defined by Puffenbarger (see Chapter 3): performance, schedule, risk and cost. The number of concepts, concept options and technology issues can be expanded relatively. The number of criteria, however, are fixed, or would at least require a significant programming effort and recoding of the pilot model to expand.

The demonstration model DSS consists of a main program and 31 subroutines (Pascal procedures and functions). It required over 3500 lines of code to meet the objectives of the pilot model. The program was written in a structured form as is encouraged by the use of the Pascal language, and it is modular in its construction. The subroutines of the problem range in size according to their function.

One external function is included with the DSS package. It is a "print screen" function that dumps (prints) the screen to the printer on command. The program is called PSC.COM and is activated by the <shift F12> key sequence of the Zenith Z-100 microcomputer. The print screen function works with any printer as it sends the screen output to the list device under the Zenith's operating system. The print screen function is common for most microcomputers, in different versions, and was thus not a restrictive factor to the portability of this implementation. In fact, the DSS was also implemented successfully on a Kaypro 2 computer, which operates under a completely different operating system than the Zenith microcomputer.

To facilitate user friendliness, the Zenith implementation uses the "batch" command option of the MS-DOS operating system. The DSS command files are stored on a 5 1/4 inch floppy diskette, and are automatically called at the system startup signal. The program execution name of the STC DSS is stored in a batch file called "autoexec.bat." All commands in that file are executed first, before the system enters the operating system environment. For the Zenith implementation, the PSC.COM file is called first to implement the DSS print screen function, then the DSS command file is called to begin operation of the pilot DSS. This method of automatic program start-up facilitates user friendliness by allowing

users access to the system immediately, without instructions that may be necessary for other microcomputer systems. This option is available for all systems that operate under the MS-DOS operating system.

As previously stated, the demonstration model DSS was constructed using modular and structured programming techniques. In fact, the DSS has effectively structured the decision process as outlined in Chapter 3 and is best described in terms of its program flow. A diagram of the STC DSS program flow is contained in Figure 8. The following discussion of the program implementation follows the program flow as outlined in Figure 8.

Identify User. The first section of the main program begins operation of the DSS by making sure that the user wants to enter the DSS. The option of immediate program termination was designed into the system due to the use of the automatic start-up feature implemented in the DSS as outlined above. If the user wishes to enter the DSS, program execution begins. The main program calls a procedure that determines the users name. The users first initial, middle initial and first six letters of the last name are used by the DSS to open a file under the users name that becomes that users private database. This file contains records of all previous runs of the DSS by that user. The information contained in each record includes:

- 1) The name of the concept worked with,
- 2) The name of the data file that contains all of the information pertaining to that concept,
- 3) All option names for this concept,
- 4) All of the pairwise comparisons made by the user,

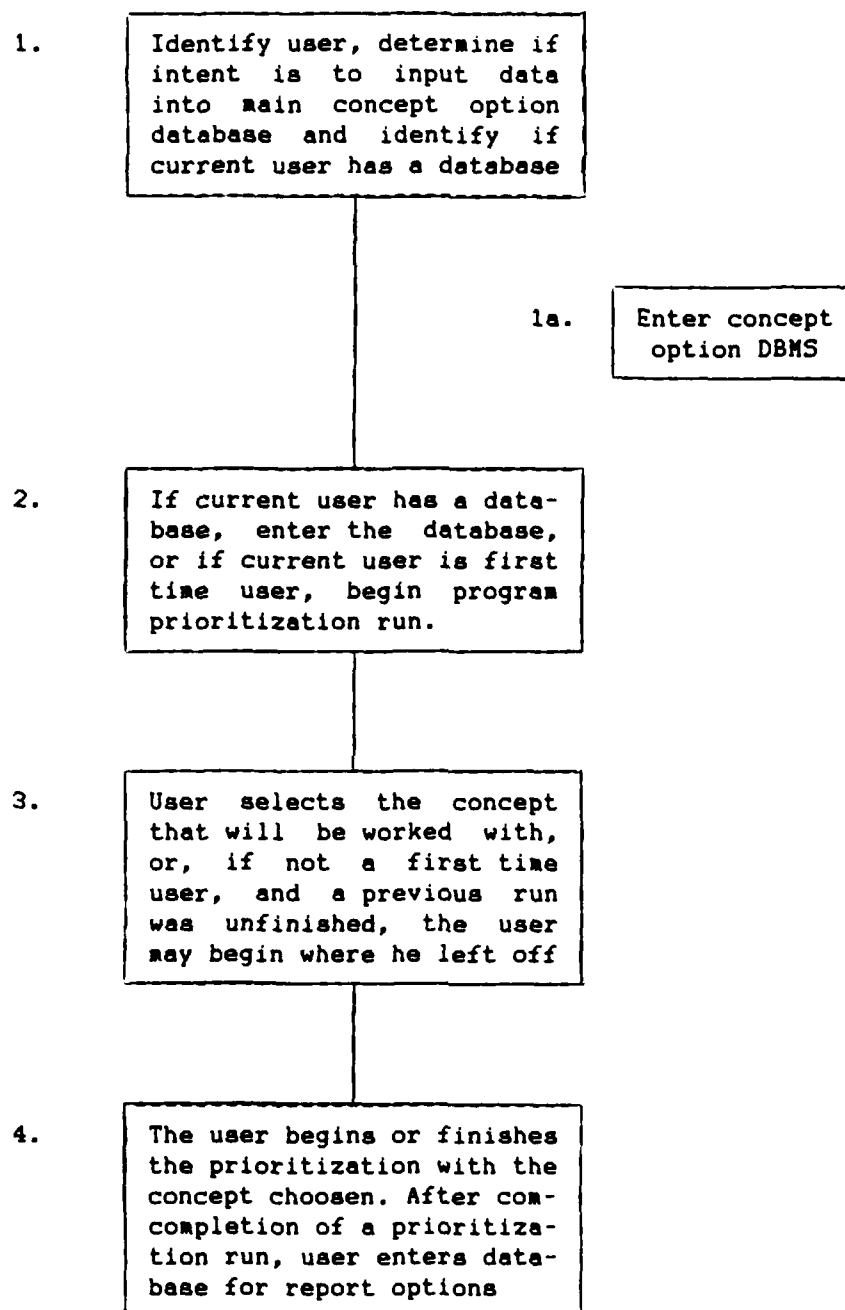


Figure 8. STC DSS Functional Flow

5) All of the final priority vectors as calculated by the AHP model,

and 6) The consistency ratios of the comparisons.

The user friendly features of this section include:

- 1) Warnings if the user states that he has not used the system before, while in fact he has, that his current database will be overwritten, if he does not change his answer,
- 2) Echoes the user name, as given, to insure that it is accurate,
- 3) Warns the user if he has input a user name for which there is no file, and states that he has used the system before.

The final decision of this section is whether the user has entered the system to input data into the concept database. The concept database is a series of ten files that contain up to five records each. There is one file per concept stored on disk. The records inside the files contain the complete set of information about the concept options for a given concept. The records in the concept files contain the following information:

- 1) The name of the option,
- 2) The originator of the option,
- 3) All of the technology issues associated with the concept option (up to 15), and the information associated with each technology issue, that is, schedule, risk and cost information,
- 4) The performance values of the concept option, based upon the six areas of coverage, capacity, quality, timeliness, availability and survivability.

Although the original intention of the pilot model was to demonstrate database management concepts through the manipulation of each individual users database, user inputs on an earlier version of the DSS

indicated that proper database management principles should be demonstrated for the concept database as described above. This input led to the inclusion of a database management module of the main program. The decision sequence, as defined above, leads the user to the database management module, described below.

Database Management. The database management capabilities of a DSS are a critical feature of a DSS (see Chapter 2). The proper demonstration of those capabilities is critical to the success of the pilot model approach (see Chapter 3). The capabilities of the STC DSS database management section are designed to maximize the education of the potential users. The process of designing and implementing the database management system was also informative to the designer and programmer (in this case, both hats were worn by the researcher) and as this was the stated intention of the pilot model the information gained will be included in this research. First, however, the capabilities of the current database management system will be reviewed.

Upon the decision of entering the database management system, the user is presented a menu that presents the available options. Figure 9 is a reproduction of this menu. The database management system allows the user to add, change, or delete all or just a portion of the data contained in a concept file. If the user chooses to enter a new concept, he is given a warning that he should enter at least two concept options, with all of the required information. Although this may seem restrictive, and it is, it is not unreasonable for the pilot model implementation. The goal of the pilot model is to show what can be done, not necessarily do everything. In this case, the extra coding effort necessary to only display concept options that had two or more

CONCEPT DATABASE MANAGEMENT SYSTEM

DATABASE OPTIONS

- 1) ENTER a new concept
- 2) CHANGE or add data to
a concept
- 3) ERASE a current concept
- 4) CONTINUE program
- 5) QUIT program

SELECTION:

Figure 9. Database management options menu.

concepts with complete information was deemed unnecessary. The pilot model DSS was working with contrived data (not real) and the entire data sets could be entered. As a final note, if the user wished to enter a new concept, and ten were already part of the system, he was given a message requiring that he delete an option before he could input another.

When the user selects the second option, to add or change the data of a concept, a second menu is generated. This menu displays all of the concepts, and asks the user to decide which concept he wishes to work with. Once a selection is made, the user is presented with a menu that gives the options available for adding or changing data at the concept option level (within a given concept). Figure 10 is a reproduction of this menu. As in the main database management section, the user may add an entire new option. If there are already five options, he is told so by the program, warned that he must delete an option before another may be added, and returned to the menu of Figure 10. If option 2 is selected, to change data, the user is presented with a menu of the

Choose the appropriate number

Input Options

-
- 1) ADD an option
 - 2) CHANGE an option
[or part of one]
 - 3) DELETE an option
 - 4) None of the above

SELECTION:

Figure 10. Concept option DBMS options

concept options. When a concept option is chosen, the user is given a menu that determines what portion of the information within the record that he wishes to change. At each stage of this process, the information that the user inputs to the system is checked to insure that it is the correct ordinal type. That is, if the system requires integers, it is getting integers. The information is at all times displayed back to the user for confirmation before it is entered into the database. If the user elects to delete an option, a menu listing the options in the database is generated and the user selects one. Before anything is deleted, the user is asked to confirm that he wishes to delete the option he has selected. If he balks, he is returned to the Figure 10 menu. Finally, selection of option 4 from the Figure 10 menu results in the program reverting back to the Figure 9 menu, the main database management menu.

Selection of the third option of the main DBMS menu (Figure 9), to delete an entire concept, functions exactly like the previously described delete function for the concept options, except on a larger scale. The user is again asked to confirm the erase decision, and if it

is negative, is returned to the Figure 9 menu.

Selection of the fourth option of the Figure 9 menu, continue the program, allows the user to enter the user database portion of the DSS or the model system of the DSS.

Finally, selection of the fifth option terminates the DSS. This option was included to account for the possibility that the persons responsible for maintaining the concept database (each user maintains his own database) do not perform any prioritization.

Some of the assumptions necessary to construct the pilot model had an effect on the design of the database and its management capabilities. For example, the construction of the database using Pascal files and records limits the expansion of the database. If new data requirements were identified, and the DSS database did not already have space available for the data, major code modifications would have to be made to accommodate the change. Also, the data already in the files would have to be transferred to the new files. This would also require a significant programming effort. The above limitations, however, do not really limit the use of the pilot model. In fact, such limitations can spur the type of interaction between the users, designers and programmers that the pilot model implementation strategy advocates.

As was the case of the entire development effort, user friendliness was the overriding consideration during construction of the database management module for the demonstration DSS. Although some of the user friendly features of this section have been mentioned earlier, they are restated here for emphasis. The user friendly features are:

- 1) Menu driven screens of choices at all levels,
- 2) Constant looping of the DSS execution to avoid

irretrievable mistakes,

- 3) When entering data into the database, the information is echoed back for approval and checked for correct type,
- 4) All mistakes enter error loops that do not allow the user to leave the loop until the mistake is corrected.

User Database. This demonstration model DSS was originally intended for demonstration to users who would not be concerned with the input of data to the database. It was, however, considered important that they be exposed to some of the database management principles as were discussed in the previous section. For this reason the pilot DSS makes an individual file for each user that serves as a private database. Each user is restricted to their own database, and is not allowed entry into the anyone else's file. Of course, if the user enters the program using someone else's name, access to that file will be gained. The information contained in each record of the user database was listed in the Identify User section.

Assuming that the user has used the DSS before, the user database is entered either upon exiting the DBMS section or directly after the introduction section (if the DBMS option is not taken). The user is immediately presented with the menu depicted in Figure 11. The first statement of the menu tells the user his current status with the system. In the case depicted above, the user had last used the system in prioritizing Spaced Based Laser concept options. He has used the DSS approximately 11 times, and did not finish the 11th program run. If the user had finished all of the prioritization process, the message at the top of the menu would indicate that situation. It should be noted that each user is allowed only one incomplete run, and this run will always be

You have 10 previously completed program runs, and
1 incomplete program run, number 11, and you were
working on Space Based Laser.

REVIEW OPTIONS

- 1) View a previous run
- 2) Print any run
- 3) Start a new run
- 4) Continue unfinished run
- 5) Erase all previous runs
- 6) Quit DSS

Please input the number of your choice:

Figure 11. User database review options menu.

stored as the very latest in the database. The first option that a user has from this menu is the VIEW option.

The VIEW option allows the user to select a record in his database and look at what he has done in the past. The screen output from the view section is the same as the DSS uses during execution to cause less confusion on the part of the user. The user, upon entering the VIEW option is asked to select the number of the record that will be viewed. The user also has the option of returning to the Figure 11 menu. Once a record is selected, the user is presented with the menu presented in Figure 12. This menu allows the user to select a section of the program to view. When a section is selected, the user is presented with the output of the record he has chosen to view. The user is asked to hit return when finished with the screen, and is returned to the Figure 12 menu. If the user is viewing an unfinished run, only the sections that have been completed can be viewed. If the user selects an unfinished

Spaced Based Radar

SECTIONS TO VIEW

-
- 1) Criteria
- 2) Performance
- 3) Schedule
- 4) Risk
- 5) Cost
- 6) Final priorities
- 7) Return to previous menu

The catagories above refer to the sections of the program in which the prioritization was performed. Select the number of the section you wish to VIEW:

Figure 12. VIEW menu, first option of review section.

section for viewing, the DSS will ask for another section automatically. When the user selects the option 7 of the Figure 12 menu, the user is asked if another record will be viewed. The user can also elect to return to the Figure 11 menu, the main user database menu.

The second option of the review section (user database) is to print out any record. Only records containing complete DSS program runs may be printed. After selection of this option, the user is asked which record is to be printed. Upon selection of a record number, the output is sent to the computers default list device. The user can then select another record to print or return to the Figure 11 menu.

The user, by selecting option 3 from the main review menu, can enter the model management portion of the DSS. This option initializes (erases) an unfinished run that the user may not want to complete, or starts a new run if all previous runs were completed.

Selection of option 4 of the Figure 11 menu allows a user with an

incomplete run to enter the modeling system at the point that it was left. If a user who does not have an incomplete run tries to select this option, a correction loop is entered until another option is selected.

Option 5 of the user database option menu is a function that erases all of a users database records, except the latest program run. Upon selection of the option, the user is asked to verify if he wishes to erase the file of his records. If the answer is affirmative, the file is erased, then reopened, with the latest run becoming the first of the new file.

The sixth and final option of the review section is to quit the DSS. Selection of this option enters the quit confirmation loop. That is, if it is selected, the user is asked to confirm that the real intention is to stop execution. If it is, the DSS execution is terminated. If not, the user is returned to the Figure 11 (review) menu.

This section of the program uses many of the same user friendly features that were mentioned in the previous sections. The system automatically error detects and traps, then enters correction loops until an acceptable answer is presented. The system also provides step-by-step menu instructions which are self-explanatory and easy to follow. Finally, the system provides for input mistakes. That is, the user can not inadvertently erase his database, as he is constantly asked to verify his intentions.

Options 3 and 4 of the review menu allow the user to enter the modeling subsection of the DSS, which is also entered automatically by a first time user (by-passing the user database review section because

there is nothing in the user database yet). The modeling section of the DSS actually incorporates blocks 3 and 4 of Figure 8, the DSS functional flow diagram. As this is the case, these blocks will be described together in the next section.

Model Management. The first stage of the modeling section is the presentation of all concepts currently contained in the database. The concepts are presented so that the user may select a concept that he wishes to work with. This stage is skipped if the current user has and is executing a previously unfinished run. The user is not allowed to leave the program at this stage.

After the user has chosen a concept, the prioritization process begins. The process, or model, is composed of five parts, as was described in Chapter 4. First, the user is presented with pairwise comparisons that help to prioritize the attributes (criteria - performance, schedule, risk or cost) based on their relative importance to this concept. Additional information that the user is presented with are the Analytic Hierarchy Process Comparison Scale (Table 3, Chapter 4) and a table of information concerning the attributes. The user is required to view the AHP scale at this stage of the program, but has the option of viewing the description of the attributes. The user, through the pairwise comparisons, ranks the four attributes. The program, without the user's knowledge, has built the Analytic Hierarchy Process (AHP) decision matrix automatically, and solved it for the prioritized weights of the criteria. The pairwise comparisons made, the priority vector of the criteria, and the consistency ratio (see Chapter 4) of the comparisons for this section are saved in the user's database, in the current record. The priority vector and the consistency ratio value are

also displayed to the user. If the user does not agree with the current priority vector, the DSS loops back to repeat the process. The DSS can continue the loop until the user is satisfied with the results. Finally, the user may quit the DSS after the criteria prioritization section is complete. The input and output values generated will be saved, and an appropriate message indicates that to the user.

After the prioritization of the criteria, the first level of the AHP decision hierarchy, the user is asked to prioritize the options for a given concept based upon a given criteria. The criteria are presented in the following order: Performance, Schedule, Risk and Cost. The user is first presented with any tables needed to clarify the values from the database and the option of reviewing the AHP comparison scale. The program then presents information from the database for the user to use in making the judgements about the options. The information concerning a concept option is presented in aggregate in all of the main screens of the four attribute sections. Three of the sections, Schedule, Risk and Cost, can break the information down into individual Technology Issues, and present it for each of the concept options. The user can elect to leave the system after the Performance, Schedule and Risk sections, and the input and output values for all comparisons made to that point are saved. All of the four subsections display the final priority vector of the users comparisons and the consistency ratio, and ask if it is acceptable. If not, each section loops until the user is satisfied.

As was mentioned above, the user is not given the option of leaving the program after comparing the options based on the Cost criteria. This is because the user has completed the prioritization process and the system automatically presents the user with the final priority

screen. This screen presents the final priorities of the options for this concept based upon the comparisons of the criteria and of the options based on the criteria that the user has made. The user is asked if this priority ranking is acceptable, and if not, the user is looped back to the beginning of the model process (prioritization of the criteria) to begin the program again. If the DSS run is acceptable, the DSS automatically saves the data in the user database and enters the review section.

The model section of the DSS also has all of the user friendly characteristics of the other sections of the program. The DSS checks for correct input and places the user into a correction loop when the input is not of the correct type or not an option. The DSS also displays many explanation screens that explain the operation of the system and the procedures required of the user.

Program Performance

System response time is considered to be a major factor in user friendliness and in gaining user acceptance for computer based implementations (TT:48). The problem of system response time contributed to the difficulty of programming the pilot DSS, but was given careful consideration throughout the development of the demonstration model.

Most of the menus and questions that the DSS exhibits are generated internally, as well as all system error messages. This means that the menus, messages and error messages are presented quickly after an input by the user. The response time is not significantly influenced by the size of the menu, and in all cases the menu is presented faster than the user can assimilate the information that is appearing on the screen.

The solution of the matrices produced by the user's pairwise comparisons (finding the principle eigenvector) takes over 10 seconds. This is a noticable difference in the response of the DSS from other sections so messages were added that indicate that the system is still working on the problem.

Finally, due to the size of the program, the object code of DSS is contained in 4 different files. That is, the program operates as a series of overlays. The main program has blank sections in the compiled object code that are used by the overlay files. The overlay files contain the object code of functions and procedures that are called one at a time, as needed, to perform system functions. By only placing the least used subprograms into the overlay files, system response time is not noticeably degraded.

Final Comments

The pilot model implementation is not and should not be a working model DSS. It must, however, display the major characteristics of a complete DSS to fulfill its objectives. This Chapter describes the implementation of the STC demonstration model DSS. The major emphasis of the design was user friendliness, as was described throughout the implementation description, while the design also incorporated the AHP decision model and database management functions. This implementation demonstrates the major characteristics of a working DSS, as was intended.

VI. Analysis of the Pilot Model DSS Implementation Strategy

The pilot model approach advanced in Chapter 3 of this research was initiated for the Space Technology Center (STC). The benefits of this implementation effort have been explicitly and implicitly stated throughout this report. This chapter summarizes those benefits and discusses insights gained from the implementation effort. First, the goals of the pilot model strategy, and how they were met, will be reviewed.

Pilot Model Implementation Goals

A major goal of the pilot model implementation approach is to educate the users of the pilot DSS about the possible capabilities so that they may better define their requirements (type and format of data, type of analysis, possible functions, type of user interface, displays, etc..). To begin this process, an early version of the STC pilot DSS was presented to several potential users at STC. Even though the first demonstration model DSS was restricted to a single pass through the implemented decision process, it caused the users at STC to begin stating requirements that they felt would be helpful to them. These comments were assimilated by the DSS designer and programmer (in this case, the researcher) and translated into the development of start and stop procedures, database access capabilities, a user controlled personal database feature, and expanded hardcopy print capabilities.

The content of the comments, however, was not as important as the type of the comments received. There was a distinct difference between the type of user comments before and after the pilot model DSS was demonstrated. When program requirements for the pilot DSS were first

researched at STC, the potential decision-makers or users were asked to identify requirements for the demonstration model. The user guidance received at that time was vague and inspecific. It was clear that STC decision-makers were asking for some type of decision aid. What was unclear was what type of aid was wanted, what it should do, what data was needed, and how it should be displayed. The decision-makers were ineffective in communicating the system requirements that they wanted for a DSS. After the first iteration of the pilot model, however, the user comments were more specific and focused. The user was able to make a comparison between the capabilities provided by the first pilot DSS and the capabilities the user wanted, but was unable to quantify.

There is evidence, then, that the pilot model does educate and stimulate the user into specifying system requirements more effectively. If this is the case, the pilot model implementation approach may be an extremely valuable tool for DSS development. Users would learn how to state their requirements before, not during development of the first operational DSS. In contrast, when using a prototype development strategy the user is learning how to state requirements during the critical development phase of the system. By using the pilot DSS, the amount of time and effort required to get the first satisfactory DSS on-line may be shortened, thus saving costs in the long run.

The second goal of the pilot model implementation strategy is to generate organizational acceptance (and possibly advocacy) and to stimulate the interest of other potential users in decision support systems. The pilot DSS accomplished this in part by the decision process that it modeled. This research was supported by a sponsor concerned with finding a prioritized list of all technology issues. Another department

at STC is concerned with finding the best satellite point design (concept option) for a given concept. The pilot DSS, as described in Chapter 5, attempts to define the best concept option based upon its unique technology issues. Although the pilot DSS does not completely accomplish either task, it demonstrates portions of both decision processes. In fact, the two processes may be linked, in that one feeds information to the other. By structuring the STC pilot DSS in this manner, at least two potential users within the organization could be interested in the operation of the pilot system. It appeared that the STC pilot DSS did generate interest from different potential users.

The measurement of user acceptance levels and the degree of advocacy within an organization is extremely subjective. Without a measure, however, it is virtually impossible to determine the success of the pilot model approach in meeting its second objective. Some method of measuring the increase or decrease of user acceptance due to the pilot model must be found. One possible alternative may be to keep records of DSS planning meetings. As more users become involved, and the level of their involvement increases, it could be assumed that the pilot model is accomplishing its goal. This form of measurement is not rigorous to any degree, and highlights the need for further research in this area.

The third goal of the pilot model implementation approach is to stimulate discussions between users, DSS designers and programmers to eliminate costly errors resulting from miscommunication. This process began at the earliest stages of the pilot model development effort. In this case, the DSS designer and the programmer are the researcher. Communication between the user and the designer/programmer led to the expanded version of the pilot model. As the user and the designer/

programmer exchanged ideas about the pilot DSS, it became steadily easier to understand the other parties point of view.

This researcher's personal experience indicates that the pilot model stimulates communication between the users, designers and programmers involved in the implementation effort. By stimulating communication of the different players involved in the development of a DSS capability, the pilot model approach may reduce costly errors. As in the first goal, the pilot model approach is geared toward producing a planning payoff. The pilot model stimulates communication between the players early, before critical design stages are accomplished. Again, this communication is occurring, if at all, during the critical design stage of a prototype approach. If this is the case, a pilot model approach has a clear advantage over prototype implementation strategies.

The fourth and final goal of the pilot model implementation strategy was to give designers and programmers an understanding of the current database. This goal was established as most decision support systems (as discussed in Chapter 2) are built around existing databases. At the time of this research STC was involved with constructing a computerized version of the MSSTP database. Although the database was not complete, its structure was known. Therefore, the categories of data elements contained in the database were also known. Earlier, in Chapter 4, it was assumed that the DSS would receive the data that it needed in the proper form directly from the MSSTP database. This was not the case, however. An excellent example is the storage of technology issue risk values. Risk values are stored as characters in the MSSTP database, and as integer values in the pilot DSS. Specifically, the values for risk from the MSSTP database are vl, l, m, h, and vh, representing

very low, low, medium, high and very high. The values in the demonstration model DSS database range from 1 to 5, with 1 indicating very low risk and 5 indicating very high risk. The numbers 2, 3, and 4 correspond to the other MSSTP database values. For the pilot DSS to use the MSSTP database, an interface module would have to be constructed that converted the characters of the database to integers for the DSS.

The personal experiences of the researcher again indicated that designing and implementing a pilot model DSS generates an understanding of the database that must be used in conjunction with the DSS. Furthermore, the pilot model stimulates thought into the topics of database management, structure and content for the pilot DSS and the complete DSS that may be developed later. The strength of this conclusion is that the pilot approach may force this type of thought earlier in the system development cycle than other implementation approaches. If this is the case, the pilot model implementation again shows the potential for avoiding costly mistakes. If problems exist in the structure of a database, a quick pilot DSS may lead to their discovery and correction faster than a prototype approach. The early consideration of database management and content needs may also reap future benefits, by providing designers/programmers with an initial idea of the magnitude of the DBMS task.

General Comments

Software selection is one of the most important factors in designing any microcomputer based program, and a pilot model DSS is no exception. The STC pilot DSS was constructed using Turbo Pascal. Pascal's advantages (as outlined in Chapter 4) include the use of structured data types. One Pascal data type, records, formed the basis

for the pilot model (system and user) database. On the other hand, Pascal records and files of records constructed for the demonstration model database are not easily changed. If new data elements were added to the requirements of a Pascal record type database, a separate program would have to be written to move the data from the old files to the new files. Procedures to accomplish this automatically from within the DSS would be impossible to construct without prior knowledge of the changes. The lack of flexibility in the database is not a handicap for the pilot model. It is, however, unacceptable in any working DSS. A record type database like the STC pilot DSS implementation is not recommended for the complete DSS. An interesting alternative is available. A database package, such as dBase II (see Chapter 4), can be used if it meets two requirements. First, it must support its own programming language. By having its own programming language, the database package can be modified to fit the needs of the DSS. Secondly, the package should be able to call programs outside of itself and be called by other programs. By supporting this requirement, the database package overcomes most of its disadvantages. The modeling system could be written in a language such as Pascal and stored on disk as a command file. An overall command program could be written that could call and control both the database package and the modeling program.

Finally, the use of a database package may yield some advantages. First, it would make the addition or subtraction of data elements much easier. The capabilities to perform these functions are already contained in the database management system and would not have to be duplicated. Secondly, the database package may support features, such as a query subsystem, that would be useful to the DSS. A good database

package that meets the two specified requirements may make an excellent vehicle for DSS development on microcomputers.

VII. Conclusions and Recommendations

The main objective of this research was to develop an effective implementation strategy for the application of decision support systems to the prioritization of space technology issues at the Air Force Space Technology Center. The pilot model implementation approach was developed to satisfy the main research objective. The objectives of the implementation, as summarized from Chapter 3, are:

- 1) The system educates the users about DSS concepts enabling them to better define their requirements,
- 2) Pilot models can generate acceptance and advocacy of DSS in an organization, and attract more users,
- 3) The approach educates users, DSS designers and programmers in the other players point of view. This reduces the chance of miscommunications,
- 4) Helps designers and programmers gain an understanding of the current database. Stimulates thought into DSS database structure, content and management.

Each of the objectives of the pilot model implementation strategy were achieved, to some extent, for the STC implementation. It was found that the pilot model played a significant role in helping the user define requirements. The pilot model provided a basis for comparison between user expectations and reality. The pilot model also served as a medium to attract more users. Because it did not have to perform as a working DSS, the demonstration model was constructed to appeal to a wide range of users. Furthermore, the pilot model served as a catalyst to stimulate feedback from each of the DSS players. Finally, the pilot model educated the designer and programmer in database management concepts. The structure, content and display of the information within the pilot DSS database generated thought about how to perform these tasks

most effectively and efficiently.

The pilot model implementation strategy has the potential of saving an organization time, effort and costs. There is a balance that must be struck, however. If the development effort of the pilot model is extensive, and the DSS approaches the capability of being used by the organization, it has defeated its own purpose. On the other hand, if the pilot model is not sophisticated enough, it may not only fail to meet its objectives, but seriously damage the organizations view of decision support systems in general. A pilot DSS that meets the objectives outlined above may be an opportunity for an organization to assess its decision support requirements at minimum cost.

Recommendations

No implementation process is self-sustaining, the pilot approach included. Although the pilot DSS has been completed, it is only the first and easiest portion of the decision support system development process. Much further work remains to be accomplished. The formation of a DSS working group at STC is the next logical step. The group, or an outside agency in conjunction with the group, must begin to define the complete STC decision process. Once the decision process is completely defined, the identification of all models, operations research techniques, database management and database needs is assessed and evaluated. The capabilities and style of the inputs and outputs to the DSS must be reconsidered in light of the knowledge gained from the pilot DSS. Finally, the DSS group must begin the selection and integration that meets the objectives established by the group.

The pilot model implementation strategy should be applied to other

organizations and decision processes. One case study does not prove the worth of an implementation process. In addition, some quantitative measures of performance should be developed to measure the success of the implementation strategy. Traditionally, performance measures for DSS have been difficult to define. This may also be the case for the pilot model implementation strategy.

BIBLIOGRAPHY

1. Alter, S.L., "Development Patterns for Decision Support Systems," MIS Quarterly, Vol. 2, No. 3, September, 1978, pp. 33-42.
2. Alter, S.L., Decision Support Systems: Current Practices and Continuing Challenges, Reading, Massachusetts: Addison-Wesley, 1980.
3. Bonczek, R.H., C.W. Holsapple, and A.B. Whinston. "The Evolving Roles of Models in Decision Support Systems," Decision Sciences, Vol. 11, No. 2, 1980, pp. 339-356.
4. "Classified Document: Qualified requestor may obtain this reference from AFIT/ENS, Wright-Patterson AFB, OH 45433."
5. Clark, Thomas D. Unnamed working paper, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1984.
6. Cooper, Doug and Michael Clancy, Oh! Pascal! New York: W.W. Norton & Company, 1982.
7. Davis, Gordon B. and Margrethe H. Olson. Management Information Systems: Conceptual Foundations, Structure, and Development. New York: McGraw-Hill Book Co., 1985.
8. Denise, Richard M. "Technology for the Executive Thinker," Datamation, Vol. 29, No. 6, June, 1985, pp. 206-216.
9. Ginzberg, M.J., and E.A. Stohr. "Decision Support Systems: Issues and Perspectives," Decision Support Systems, New York: North-Holland Publishing Co., 1982, pp. 9-31.
10. Gorry, G.A., and M.S. Scott Morton. "A Framework for Management Information Systems," Sloan Management Review, Vol. 13, No. 1, Fall 1971, pp. 55-70.
11. Henderson, J.C., and R.S. Ingraham, "Prototyping for DSS: A Critical Appraisal," Decision Support Systems, New York: North-Holland Publishing Co., 1982, pp. 79-96.
12. Jones, Steven T., "Programming: From BASIC to Turbocharged," Profiles, Vol. 3, No. 3, October 1985, pp. 24-31.
13. Keen, P.G.W. and Michael S. Scott Morton. Decision Support Systems: An Organizational Perspective. Reading, Mass: Addison-Wesley Publishing Co. 1978.
14. Keen, P.G.W., "Adaptive Design for Decision Support Systems," Data Base, Vol. 12, Nos. 1 and 2, Fall 1980.
15. Keen, P.G.W., "Decision Support Systems: Translating Analytic Techniques into Useful Tools," Sloan Management Review. Spring, 1980. pp. 33-44.

16. Koble, Roger D., Research on Applications of Computers as an Aid in Decision-Making in Air Force System Program Offices, Unpublished Master's Thesis, AFIT/LS Wright-Patterson AFB, OH. September, 1985.
17. Little, J.D.C., "Models and Managers: The Concept of a Decision Calculus," Management Science, Vol. 16, No. 8, April 1970, pp. B466-B485.
18. Moore, J.H., and M.G. Chang. "Design of Decision Support Systems," Data Base, Vol. 12, Nos. 1 and 2 (Fall 1980), pp. 8-14.
19. Nobles, Clayton M., A Management Information System for the SAC OPSEC Program, Unpublished Master's Thesis, AFIT/LS, Wright-Patterson AFB, OH, March, 1985.
20. Puffenbarger, John, A Methodology for Assessing Technology Trade-offs of Space Based Radar Concepts, Unpublished Master's Thesis, AFIT/LS, Wright-Patterson AFB, OH, December 1985.
21. Rensema, Peter H. and Randall W. Chapman, A Decision Support Methodology for Space Technology Advocacy. Unpublished Master's thesis. AFIT/LS, Wright-Patterson AFB, OH, December 1984.
22. Saaty, Thomas L., "Priority Setting in Complex Problems." IEEE Transactions on Engineering Management, Vol. Em-30, No. 3, August, 1983., pp. 140-155.
23. Saaty, Thomas L., The Analytic Hierarchy Process. New York: McGraw-Hill, 1980.
24. Savvy User Manual, Excalibur Technologies Corporation, Albuquerque, New Mexico, 1984.
25. Simon, Herbert A. The New Science of Management Decision. New York: Harper and Row, 1960.
26. Spector, Bertram I., "Decision Support Systems," News From the DSS Practice of Booz-Allen & Hamilton Inc. Bethesda, Maryland: Booz-Allen & Hamilton Inc.
27. Sprague, Ralph H., Jr., and Eric D. Carlson. Building Effective Decision Support Systems. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.
28. Strang, Gilbert, Linear Algebra and its Applications. London: Academic Press, 1976.
29. Thierauf, Robert J., Decision Support Systems for Effective Planning and Control. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.
30. White, Greg R., Personal Computer Aided Decision Analysis, Unpublished Master's Thesis, AFIT/LS, Wright-Patterson AFB, OH, December 1984.

APPENDIX A

Verification of Computational Algorithms

Square Root

The square root function was validated as it was a major contributor to the calculation of the maximum eigenvalue. The maximum eigenvalue of the matrix of comparisons determined the priorities of the elements that were compared. To verify the square root function, the output from the Turbo Pascal square root function was tested against the output of MBASIC's (Microsoft BASIC) square root function. The argument of the function was stepped from 0.05 to 10.00 in increments of 0.05. The output from both programs was carried to eight significant digits. The results of the Turbo Pascal square root function were equal to the values of the MBASIC square root function. The Turbo Pascal function was accepted as valid.

The verification check was limited to the range 0.05 to 10 because this range is approximately the limit for which the square root function is called in the AHP prioritization procedure.

Mean and Standard Deviation

The pilot DSS was required to calculate mean and standard deviation information throughout the modeling subsection. To verify the mean and standard deviation calculations yielded correct results, the formulas used were checked by hand calculator. The mean was defined as the sum (Σ) of n elements divided by n . The standard deviation was defined as the square root of the sum of each element minus the mean. The results of the hand calculations matched the output of the pilot DSS for five trial runs. The computation of the means and standard deviations were considered valid.

Priority Vector and Consistency Ratio

The priorities and consistency ratios found by the pilot DSS were verified by the comparison of those values to results published by Saaty (YY) for an example problem. The output of the demonstration model DSS matched the values presented by Saaty. The output from the pilot DSS was considered valid.

The priority vector calculated above was based on the maximum eigenvalue of the comparison matrix. The method used to find the maximum eigenvector is called the ordinary power method. The ordinary power method operates on the principle of a difference equation. It begins by guessing an initial eigenvector for the matrix and then successively forms new estimates. The method is essentially taking a limit (28:274). This method is not computationally efficient. The lack of efficiency, however, only effects large matrices. As the pilot DSS works with matrices no larger than 5 X 5, the relative inefficiency of the power method can be overlooked. A final point concerns the number of iterations needed to accurately estimate the eigenvalue. Strang recommends at least 20 iterations (28:274). The pilot model DSS uses 50 iterations to insure accuracy. There is no loss of program speed because the matrices the DSS works with are small.

APPENDIX B:

SPACE TECHNOLOGY CENTER PILOT MODEL

DECISION SUPPORT SYSTEM

USER'S GUIDE

CONTENTS

I	INTRODUCTION	75
	Decision Support Systems	75
	STCPMDSS	75
	Structure of Manual	76
II	USING THE PILOT MODEL DSS	77
	Current Implementation	77
	Starting the STCPMDSS	77
	System Use	77
	Comments	78
III	Program Specifics	79
	Software Environment	79
	Program Source Code	80
	Disk Files	81
	Implementation on Other Microcomputers	82

I. INTRODUCTION

This document is a reference manual for the Space Technology Center Pilot Model Decision Support System (STCPMDSS) for prioritizing space technology issues. It assumes a basic knowledge of the Military Space Systems Technology Plan.

Decision Support Systems

A decision support system is a computer-based information system that aids a decision-maker in solving complex and unstructured problems. The computer is used to provide structure to parts of a decision process while leaving the flexibility of manager control and judgement.

STCPMDSS

The STC Pilot Model Decision Support System is a demonstration model decision support system designed to display some of the most important characteristics of a working DSS. The purpose of the pilot model is to provide a basis for understanding the nature and characteristics of a decision support system within the STC decision process. The major characteristics that the pilot model displays are user friendliness, model management and database management. The objectives of the pilot model are: 1) To educate users about potential DSS capabilities to provide a frame of reference for a more effective statement of requirements; 2) To generate acceptance and possibly advocacy of DSS concepts within the target organization; 3) Give users, programmers, and DSS designers an appreciation of the others knowledge requirements; and 4) To educate designers and programmers an understanding of the current database related to the information requirements of the proposed DSS database.

In short, the pilot model DSS is meant as a quick and dirty demonstration to educate all potential players in DSS development about DSS concepts. The pilot model is not meant to be used in any capacity other than as a demonstration. It hopefully reduces long run costs by stopping miscommunication errors.

Structure of this Manual

This manual is intended as a reference for any individual concerned with either the use, design or programming of a DSS for the prioritization of space technology issues at the Space Technology Center. The manual is broken into two main sections.

The first section of the manual describes how to use the STCPMDSS. It describes how to execute and use the demonstration model. This section is presented for the potential user of the STCPMDSS.

The second section of the manual describes information that pertains to programmers and designers of decision support systems. It contains information concerning the construction of the DSS software.

This manual is by no means complete. It does not provide a complete description of the program operation. In fact, no such description is intended. The operation of the pilot model should be straight forward. If it is not, the problems that occur can lead to comments from the users to the designers and programmers, as intended.

II. USING THE PILOT MODEL DSS

This section of the user's manual describes how to use the STCPMDSS software. It provides details on software capability, system files and directions for use of the system.

Current Implementation

The pilot model decision support system is currently configured for a Zenith Z-100 microcomputer system. If your computer is not a Zenith Z-100, see Chapter III of this manual for installation instructions for other microcomputer systems.

Starting the STCPMDSS

The STCPMDSS is stored on a 5 1/4 inch floppy diskette. After the computer has been turned on, program execution is initiated by inserting the diskette containing the STCPMDSS into drive A. The DSS is a turn-key system. That is, it automatically installs itself in computer memory when the drive door is closed. The STCPMDSS immediately asks the user to confirm that the program should run. If the answer is negative, program execution is terminated. Otherwise, the STCPMDSS begins program execution.

System Use

The STCPMDSS is designed to lead the user through a structured decision process that helps to define the best option (system design) for a given space system concept. The program provides menus and displays at all stages of execution. The program is designed to be completely user friendly. That is, it is designed to provide constant error trapping to guard against any unintended instruction from the

user. Furthermore, the system was designed to be self-explanatory during execution.

The best way to become familiar with the program and its capabilities is to use the system. The data provided with this implementation was contrived for example purposes only. The user may delete or change all or part of any information listed in the concept database.

The STCPMDSS constructs a user database for each decision-maker that uses the system. This database is under the control of the user. The data saved is a record of a decision-makers pass through the model subsystem of the DSS. The system automatically keeps track of the status of the system.

Comments

One objective of a pilot model DSS is to provide an example of a working DSS to help users better define requirements for a true DSS. While using the pilot model decision-makers should list their likes and dislikes of the implementation. The list may include:

- 1) Input and Output format of the screen displays;
- 2) Length of time required for program execution;
- 3) Other solution techniques that may be helpful;
- 4) Information requirements - more, less, presentation of, etc...

III. Program Specifics

This section describes the software system of the pilot model DSS. The program is not described to the level of detail necessary to make changes or modifications easily. The STCPMDSS is intended as a demonstration model only. Although it was programmed in a manner to facilitate additions and modifications, it was not meant for follow on development. A pilot model is only a frame of reference for a true working DSS that must still be constructed.

Software Environment

The STCPMDSS was written in the Pascal language. Specifically, Turbo Pascal by Borland International. No implementation specific features of Turbo Pascal were utilized. That is, the program may be compiled on any computer, using any operating system, supporting Turbo Pascal. The specific implementation of the STCPMDSS configured for the Zenith Z-100 does use two features that are machine specific.

The first implementation specific feature of the Z-100 implementation is the use of the batch command capability of the MS-DOS operating environment. The file "autoexec.bat" is called when the operating system is booted from the DSS execution disk. The commands in that file are carried out first after the computer reads the operating system. This batch file contains two commands, PSC, and DSS. The PSC command is executed first, to install the DSS print screen function.

The second Z-100 specific implementation feature is the print screen function. The print screen function for the Z-100 is a public domain command file titled PSC.COM. This file prints the Z-100 screen to its list device. That is, any printer can be used with this program.

The key sequence used to activate the print screen function is <shift F12>.

Program Source Code

The STCPMDSS was programmed in 32 modules, one main program and 31 Pascal procedures and functions. All of the program modules were coded using a consistent style.

The STCPMDSS source code is actually contained in six separate files. The source code was broken into the six files because of the limitations of the Turbo Pascal Compiler. The compiler can not handle all of the source code at once because of its buffer size. This, however, did not limit the size or capabilities of the STCPMDSS. Turbo Pascal allows for programs larger than the buffer size by placing procedures in "include" files. The procedures or functions in the include file are compiled into the main program code when the Turbo Pascal compiler encounters an include file command in the code. The include file command is a comment in the Pascal source code that tells the compiler to compile the Pascal source code in the named file at this point. The files that contain Pascal Source code for the STCPMDSS are:

- 1) DSS.PAS - main Pascal source code file;
- 2) EXTRA1.DSS - first include file;
- 3) EXTRA2.DSS - second include file;
- 4) EXTRA3.DSS - third include file;
- 5) EXTRA4.DSS - fourth include file;
- 6) EXTRA5.DSS - fifth include file.

The exact contents of the include files are unimportant, as long as normal precedence rules for Pascal procedures and functions are observed. For example, if the include files are compiled in order from 1 to 5, a procedure in EXTRA2.DSS can not call a procedure in EXTRA5.DSS.

Disk Files

Compiled Code. There were four STCPMDSS object code files compiled from the source code. The four files were:

- 1) DSS.COM - main DSS command file;
- 2) DSS.000 - first overlay file;
- 3) DSS.001 - second overlay file;
- 4) DSS.002 - third overlay file.

The three overlay files contain object code from procedures and functions preceded by the Turbo Pascal reserved word "overlay". Overlays were used in the program to reduce the size of the main execution block. Otherwise, the Pascal code would require modifications to compile it for a different microcomputer system. To make overlays, the compiler reads all the procedures and functions preceded by the word "overlay" and stores them in a file. It then leaves a space in the main object code large enough to fit the largest program in the overlay. The main program then uses that space for the procedures in the overlay file, as they are needed.

Database Files. The database structure of the STCPMDSS is a series of record files. There are two distinct databases used by the STCPMDSS. The first is the concept database. The concept database can consist of up to ten files. Each file could contain five concept options with up to fifteen technology issues per concept option. The ten files are named CONCEPT*.DSS, where the asterick stands for a letter from A to J. The number of files indicates the number of concepts contained in the database. When a new concept is added, it receives its filename sequentially.

The concept names are contained in a file called CONCEPTS.DSS, and are related to their respective concept option data files by the STCPMDSS program.

The second database is a database constructed by the program for each individual user. Upon entering the program, each user is asked for his first and middle initial and the first six letters of his last name. The DSS then opens a file using that name and a <.DSS> identifier after it.

A user file is opened each time a user gives his name upon entering the STCPMDSS. The user file contains records of the decision-makers past decision runs through the model sub-section of the pilot DSS. The user file data base is under the control of the its individual user, and is inaccessible to other users.

Information File. The final file necessary for the operation of the STCPMDSS is a file called INFO.DSS. This file contains all of the menus, tables and displays that are not generated within the STCPMDSS program code.

Implementation on Other Microcomputers

To implement the STCPMDSS on another microcomputer, a version of Turbo Pascal for that microcomputer must be available. Also, a transfer program to copy programs from one operating system to another must be found. Assuming that both of these conditions can be met, the STCPMDSS can be implemented on almost all microcomputer systems.

Procedure. First, all of the files associated with the STCPMDSS must be transferred, except for the object code files (DSS.COM, DSS.000, etc..). The source code must be transferred so that it can be re-compiled under the new operating system. Also, all of the concept database files must be transferred. Most operating systems offer programs that will mass copy files. By first copying DSS.PAS, and then copying *.DSS, all files needed for the new implementation can be

transferred.

Once the files are transferred, the Turbo Pascal compiler can be used to generate new object code files.

Limitations. The new implementation will be limited by the lack of a print screen function, unless a print screen program available for that microcomputer system. Print screen programs are available in the public domain for most microcomputer systems. Another limitation will be the lack of the automatic start up provided by the batch command function of MS-DOS. That is, if the new microcomputer does not operate under the MS-DOS (or related) operating system. If it does not, the user will have to execute a print screen function command first, and then the DSS.COM file.

VITA

1st Lt. Bruce G. Schinelli was born 28 June 1960 in Franklin, Pennsylvania. He graduated from Boonton High School, Boonton, New Jersey in 1978 and attended the United States Air Force Academy. He graduated from the Academy in 1982, with a Bachelor of Science degree in Economics. Upon graduation, he received a regular commission in the USAF, and was initially stationed at Nellis AFB, Nevada. He entered the School of Engineering, Air Force Institute of Technology, in June 1984. He is a member of Omega Rho.

Lt. Schinelli is married to the former Cecily Zahorian of Lincoln Park, N.J.

Permanent Address: 415 Liberty St.
Boonton, N.J.
07005

APPENDIX C
PILOT MODEL DECISION SUPPORT SYSTEM
SOURCE CODE


```

const maxnumconcepts = 10;  {Maximum number of concepts}
    maxnumoptions = 5;      {Maximum number of concept options}
    b = '                  '; {Blanks, for printing files}

type Char80      = string[80];  {80 character string}
    Char20       = string[20];  {20 character string}
    Char12       = string[12];  {12 character string}

outputdatafile = array[1..20] of char; {stores choices}
outputfilename = array[1..12] of char;

vector         = array[1..maxnumoptions] of real;

matrix         = array[1..maxnumoptions,1..maxnumoptions]
                of real;

alloptionnames = array[1..maxnumoptions] of Char20;

techissues = record
    tiname: Char80;
    tischedule: integer;
    tirisk: integer;
    ticost: real
end;

{techissues - A record to store the achedule, risk and cost
information associated with a technology.}

tecarray = array[1..15] of techissues;

{tecarray - an array that specifies the maximum number of
technology issues for each concept option.}

performance = record
    coverage: integer;
    capacity: integer;
    quality: integer;
    timeliness: integer;
    availability: integer;
    survivability: integer
end;

{performance - A record used to store the performance values
of a concept option.}

conceptoption = record
    optionname: Char20;
    optionorigin: Char20;
    optionti: tecarray;
    optionperformance: performance
end;

```

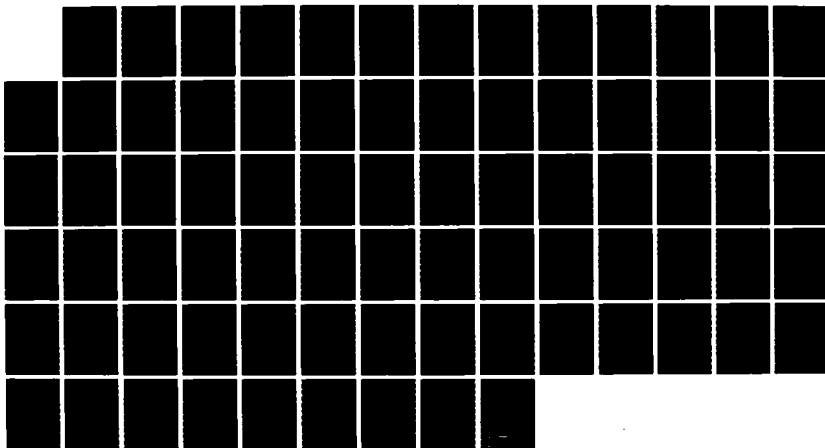
AD-A172 379

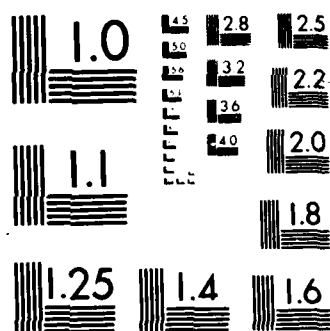
A DECISION SUPPORT SYSTEM FOR SPACE TECHNOLOGY
TRADEOFFS: A MICROCOMPUTER. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. B G SCHINELLI
13 DEC 85 AFIT/GOR/OS/85D-17 F/G 12/2

2/2

UNCLASSIFIED

NL





(conceptoption - A record that contains all of the information used by the DSS about a particular concept option. This record is the primary building block of the files contained in the database.

```
datafile = record
    conceptname: Char20;
    conceptorigin: Char20
end;
```

(datafile - type of records contained in the "concepts.dss file. used to store the name and originator of a concept.)

```
choices = record
    criteriachoices: outputdatafile;
    performancechoices: outputdatafile;
    schedulechoices: outputdatafile;
    riskchoices: outputdatafile;
    costchoices: outputdatafile
end;
```

(choices - A record used to store the choices a decision-maker makes at each stage of the AHP process.)

```
priorities = record
    criteriapriorities: vector;
    performancevector: vector;
    schedulevector: vector;
    riskvector: vector;
    costvector: vector;
    finalpriorities: vector
end;
```

(priorities - A record of all of the priorities calculated by the modeling section for each level of the AHP hierarchy.)

```
userfile = record
    conceptname: Char20;
    optiondata: Char12;
    options: alloptionnames;
    judgements: choices;
    priorityvectors: priorities;
    CRvector: vector
end;
```

(userfile - A record of all information concerning a program run for an individual user. Primary building block of the user database file.)

(In general, The type identifier is used to construct all new variable types used in the program. The type declaration for the entire program is accomplished in the main program, as type declarations are global to the entire program.)

```

Var i, quitindicator, status: integer;
    concept: Char20;
    filename, personalfile: Char12;
    answer: char;

```

(Variables - for the main program:

```

    i: An integer counting variable.
quitindicator: An integer used to govern program execution.
                If it is a 1, the program is terminated.

    status: Used to identify the status of the user.
    concept: Store the name of a concept.
    filename: Used to store the name of the file that contains
               the option information for a concept.
    personalfile: Used to store the name of a users personal database
                  file.
    answer: A character variable that is used to receive answers
            to questions by the DSS.)

```

```

(*****
*****
****                                ****
****      PROCEDURE DECLARATION OF PROGRAM DSS      ****
****                                ****
*****
*****)

```

(\$I extra1.dss) (Compile code in file extra1.dss now.)

```

procedure CheckYorN (Var check: char);

```

(CheckYorN - A procedure that insures the answer to a yes or no question is a y or n (can be capitilized). If not the program enters an error correction loop until it gets a yes or no answer.)

```

Var i: integer;

begin
    writeln;
    while not (check in ['Y','y','N','n']) do
        begin
            write ('      Please enter a "Y" ');
            write ('or "y" for yes or a "N" or "n" for no: ');
            readln (check)
        end (while check not Y or N)
    end; (CheckYorN)

```

```

($I extra2.dss) (The statements at right are Turbo Pascal )
($I extra3.dss) (command statements that tell the compiler)
($I extra4.dss) (to compile the code contained in the file)
($I extra5.dss) (named after the --$I-- sequence.      )

```

```

(*****
*****
****
****          MAIN PROGRAM BEGINS          ****
****
*****
*****)

begin

(***** Initialization *****)

  ClrScr;
  personalfile := '          ';
  quitindicator := 0;
  status := 1000;
  filename := '          ';
  concept := '          ';

(***** End Initialization *****)

(Determine if a user wishes to use the program or has entered by
mistake.)

  for i := 1 to 10 do writeln;
  writeln(b,' Do you wish to execute the AFSTC DSS program? ');
  write(b,' Please answer Y for yes or N for No: ');
  readln(answer);
  CheckYorN(answer);
  if (answer in ['N','n']) then GoTo 1;

  GetInfo(156,174); (List the title page of the DSS)
  writeln;
  write(' Press RETURN to continue: ');
  readln(answer);

(Identify the user and get his status)
  GetUserFileName(personalfile,status,quitindicator);

(This section (GetUserfile Name) allows entrance into the main
database section if it is desired.)

  if quitindicator = 1 then GoTo 1;
2:InitializeUserFile(personalfile,status);
  if status > 0 then Review(personalfile,status,quitindicator);

( ***** Review - User Database section ***** )
( review allows a user to enter his personal database. )

  if quitindicator = 1 then GoTo 1;
  if status >= 1 then GoTo 4;

( status determines where a user may go. )

```

```

(*****
(***** Beginning of Modeling subsection *****)
(*****

    ListConcepts(concept, filename, quitindicator);
(Listconcepts provides choices of concepts to work on)

    if quitindicator = 1 then GoTo 1;
3:Criteria(concept,personalfile,quitindicator);
(Criteria finds the priorities of the performance, cost, schedule
and risk sections.)

    if quitindicator = 1 then GoTo 1;
4:PrioritizeOptions(personalfile,quitindicator,status);
(Prioritizes the options according to the criteria.)

    if quitindicator = 1 then GoTo 1;
    FinalOptionVector(quitindicator,status,personalfile);
(FinalOptionVector synthesizes the final concept option priorities
from the information obtained in the other procedures.)

    if status = 7 then
    begin
        status := 0;      (User is unsatisfied with results of the)
        GoTo 3            (prioritization process, wants to repeat it)
    end (then)
    else
    begin
        status := 10;     (User does not want to repeat, is sent to)
        GoTo 2            (the review section to view or print the )
    end; (else)           (the results from the modeling section. )
1:writeln
end. (main program end)

```



```

(*****
*****      Source Code file: extra1.dss      *****
*****)

```

```

overlay function Exist(filename: Char12): boolean;

```

```

(Exist - Determines if a file on the logged disk drive of the
filename <filename> exists. Returns a boolean answer
true or false, when called. Object code for this
procedure is place in the file dss.000 because of the
overlay in front of the function declaration.

```

```

    fil: a variable indicating a file of any type.)

```

```

Var fil: file;

```

```

begin

```

```

    Assign(fil,filename);

```

```

    ($I-)                (Shuts off Turbo Pascal automatic error)

```

```

    Reset(fil);          (checking)

```

```

    ($I+)                (turns automatic checking back on.)

```

```

    Exist := (IOresult = 0) (Turbo function tha returns a zero if)
                        (an input/output error exists)

```

```

end; (function Exist)

```

```

overlay function StatusCheck(Var
                             priorityvectors: priorities): integer;

```

```

(StatusCheck - An integer function that receives a record of real
vectors and derermine the status by seeing which
vectors contain zeros. Object code for this function
is also contained in dss.000.

```

```

    i: a counter variable)

```

```

label 1;

```

```

Var i: integer;

```

```

begin

```

```

    i := 0;

```

```

(Checks the vectors for a zero to see which section a User quit at.)

```

```

    with priorityvectors do

```

```

        begin

```

```

            if criteriapriorities[1] <> 0.0 then i := 1;

```

```

            if performancevector[1] <> 0.0 then i := 2;

```

```

            if schedulevector[1] <> 0.0 then i := 3;

```

```

            if riskvector[1] <> 0.0 then i := 4;

```

```

            if costvector[1] <> 0.0 then i := 5

```

```

        end; (with priorityvectors)

```

```

    StatusCheck := i

```

```

end; (function check status)

```

overlay procedure GetInfo(firstline, lastline: integer);

(GetInfo - Enters a text file that contains most of the tables and messages for use within the DSS. It takes a first line number and a last line number and then prints everything inbetween those two numbers onto the screen.
The file that this procedure uses for this implementation is called "info.dss".

currentline: A variable used to keep track of the current line of the textfile.

letter: A character variable.

textfile: a file of text.)

Var currentline: integer;
letter: char;
textfile: text;

begin
ClrScr;
currentline := 1;
Assign(textfile, 'info.dss');
Reset(textfile);

(Advance textfile to the first line to be printed)

while currentline <> firstline do
begin
readln(textfile);
currentline := currentline + 1
end;

(while not the last line to be printed)

while currentline <> lastline do
begin

(while not the end of the line, print each character on the line)

while not eoln(textfile) do
begin
read(textfile, letter);
write(letter)
end;
writeln;

(read next line of text, advance the current line)

readln(textfile);
currentline := currentline + 1
end

end; (GetInfo)

```

overlay procedure InitializeConceptOption(Var optionrec:
                                         conceptoption);

```

(InitializeConceptOption - Takes a record of type conceptoption and initializes it by filling it with blanks and zeros, where applicable. Object code for this procedure is in dss.000.)

```

begin
  with optionrec do
    begin
      optionname := '';
      optionorigin := '';
      for i := 1 to 15 do
        begin
          optionti[i].tiname := '';
          optionti[i].tischedule := 0;
          optionti[i].tirisk := 0;
          optionti[i].ticost := 0.0
        end; (if to blank optionti)
      with optionperformance do
        begin
          coverage := 0;
          capacity := 0;
          quality := 0;
          timeliness := 0;
          availability := 0;
          survivability := 0
        end (with optionperformance)
      end (with optionrec)
    end; (InitializeConceptOption)

```

```

overlay procedure InitializeUserFile(userfilename: Char12;
                                     Var status: integer);

```

(InitializeUserFile - Receives a filename of a user database file, and a status. Returns either a new status or a blank record at the last entry position of a user database file. Object code for this file is in das.000.

userdata - a file of records of type userfile.
 userrecord - One record of type userfile.
 i,j - counting integers.)

```

label 1;

```

```

Var userdata: file of userfile;
    userrecord: userfile;
    i,j: integer;

```

```

begin
  Assign(userdata,userfilename);
  if status = 7 then
    begin
      Reset(userdata);
      Seek(userdata,FileSize(userdata)-1)
    end;
  if status = 6 then
    begin
      Reset(userdata);
      Seek(userdata,FileSize(userdata))
    end;
  if status = 10 then
    begin
      Reset(userdata);
      Seek(userdata,FileSize(userdata)-1);
      Read(userdata,userrecord);
      status := StatusCheck(userrecord.priorityvectors);
      GoTo 1
    end;    (if status is 10)
  if status = 0 then Rewrite(userdata);
  with userrecord do
    begin
      conceptname := ' ';
      optiondata := ' ';
      for i := 1 to maxnumoptions do options[i] := ' ';
      with judgements do
        begin
          for i := 1 to 12 do
            begin
              criteriachoices[i] := ' ';
              performancechoices[i] := ' ';
              schedulechoices[i] := ' ';
              riskchoices[i] := ' ';
              costchoices[i] := ' '
            end (for i one to twelve)
          end; (with judgement)
        with priorityvectors do
          begin
            for i := 1 to maxnumoptions do
              begin
                criteriapriorities[i] := 0.0;
                performancevector[i] := 0.0;
                schedulevector[i] := 0.0;
                riskvector[i] := 0.0;
                costvector[i] := 0.0;
                finalpriorities[i] := 0.0
              end (for i one to maxnumoptions)
            end; (with priorityvectors)
          for i := 1 to maxnumoptions do CRvector[i] := 0.0
          end; (with userrecord)
        Write(userdata,userrecord);
      1:Close(userdata)
    end; (InitializeUserFile)

```

```

overlay procedure Compare(Var comparisons: matrix;
                          Var A1,A2,Q1,Q2: char; i,j: integer);

```

(Compare - Makes the comparisons to build the comparison matrix for the first level of the decision process hierarchy. That is, it presents the criteria for the comparison to the decision-maker. Receives the matrix that will be the matrix used by the AHP subroutine. Also receives the letters of the comparison (A1,A2), the answer input by the user (Q1,Q2) and two integers to indicate which comparison is being made. (i,j). Object code is in das.000.

Qint: integer transfer variable.
k: counting variable.)

```

Var Qint,k: integer;

```

```

begin
  Qint := ord(Q2) - 48;
  k := 1;
  if ord(Q1) in [(ord(A1)+32),(ord(A2)+32)]
    then Q1 := chr(ord(Q1)-32);
  while (Q1<>A1) and (Q1<>A2) or (Qint<1) or (Qint>9) do
    begin
      if k = 3 then
        begin
          writeln;
          write('      Remember, the first letter of the ');
          writeln('dominating criteria first, then');
          write('      the number representing the');
          writeln('relationship from the comparison scale. ');
          write('      Ex: P5 (performance dominates ');
          writeln('compairison)');
          end;
        write(' Again?: ');
        read(Q1,Q2);
        Qint := ord(Q2) - 48;
        if ord(Q1) in [(ord(A1)+32),(ord(A2)+32)]
          then Q1 := chr(ord(Q1)-32);

        k := k + 1
        end;
      if k > 1 then writeln;
      if Q1 = A1 then
        begin
          comparisons[i,j] := Qint;
          comparisons[j,i] := 1/Qint
        end
      else
        begin
          comparisons[i,j] := 1/Qint;
          comparisons[j,i] := Qint
        end
      end;
    end; (Compare)

```

```

overlay procedure AHP(squarematrix: matrix; Var priorities: vector;
                     n: integer; Var cr: real);

```

(AHP - This procedure calculates the maximum eigen value and its characteristic eigenvector using the ordinary power method. From the eigenvalue, it calculates the priority vector of the matrix of comparisons passed by squarematrix, and returns the priorityvector as priorities. It also uses n, the number of comparisons and calculates cr, the consistency ratio. Object code is stored in dsa.000.

```

variables - i,j,m: integer counting variables.
            sum: real value sum of addition in power method.
            transform: vector that stores the characteristic
                    eigenvector.
            maxlambda: the estimate of the maximum eigenvalue.)

```

```

Var i,j,m: integer;
    sum: real;
    transform: vector;
    maxlambda: real;

begin
    sum := 0;
    cr := 0;
    for i := 1 to n do priorities[i] := 1;
    m := 1;
    while m <= 50 do
        begin
            for i := 1 to n do
                begin
                    transform[i] := 0;
                    for j := 1 to n do transform[i] := squarematrix[i,j]
                        * priorities[j] + transform[i];
                end;
            maxlambda := 0.0;
            for i := 1 to n do maxlambda := maxlambda + transform[i]
                * transform[i];
            maxlambda := sqrt(maxlambda);
            for i := 1 to n do priorities[i] := transform[i]/maxlambda;
            if m = 5 then
                begin
                    writeln;
                    writeln(b,'CALCULATING PRIORITIES');
                end;
            if m = 25 then writeln(b,'CONTINUING CALCULATIONS');
            m := m + 1;
        end; (while loop to calculate eigenvalue and vector)
    cr := (maxlambda - n)/(n - 1);
    case n of
        1: cr := 0.00;
        2: cr := 0.00;
        3: cr := cr/0.58;
    end;

```

```

4: cr := cr/0.90;
5: cr := cr/1.12;
6: cr := cr/1.24;
7: cr := cr/1.32;
8: cr := cr/1.41;
9: cr := cr/1.45;
10: cr := cr/1.49
end; (case n of 1 to 10)
for i := 1 to n do sum := sum + priorities[i];
for i := 1 to n do priorities[i] := priorities[i]/sum
end; (AHP)

```

```

overlay procedure OptionComparisons(Var optionmatrix: matrix;
numoptions: integer; Var choicevector: outputdatafile);

```

(OptionComparisons - procedure that performs comparisons between two options. Compares them by number, determined by the number of options passed by numoptions. Other variable passed to this procedure are optionmatrix, the matrix built for the AHP subroutine, and choicevector, the vector of that stores the a record of the users choices for further reference. The object code for this procedure is stored in dss.000.

Variables - i,j,k: counter variables.
Q1,Q2: character answers from input the user.)

```

Var i,j,k,I1,I2,counter: integer;
Q1,Q2: char;

```

```

begin
counter := 1;
for i := 1 to numoptions do
begin
for j := 1 to numoptions do optionmatrix[i,j] := 1
end;
for i := 1 to (numoptions - 1) do
begin
for j := (i + 1) to numoptions do
begin
write('      Option ',i,' vs Option ',j,' : ');
read(Q1,Q2);
I1 := ord(Q1) - 48;
I2 := ord(Q2) - 48;
k := 1;
while (I1 <> i) and (I1 <> j) or (I2 < 1) or (I2 > 9)
do begin
if k =3 then
begin
writeln;
write('      Remember - The # of the dominating ');
writeln('option first then the value from');
write('      the comparison scale! Ex: 17 - ');
writeln('option 1, strongly dominates, etc..');

```

```

        write('          Option ',i,' vs Option ',j,' : ');
        k := 1
    end;    (if k = 3)
write(' Again?: ');
read(Q1,Q2);
I1 := ord(Q1) - 48;
I2 := ord(Q2) - 48;
k := k + 1
end;    (while loop)
if counter in [2,4,6,8,10] then writeln;
if I1 = i then
begin
    optionmatrix[i,j] := I2;
    optionmatrix[j,i] := 1/I2
end
else
begin
    optionmatrix[i,j] := 1/I2;
    optionmatrix[j,i] := I2
end;    (if I1 = i)
case counter of
1: begin
    choicevector[1] := Q1;
    choicevector[2] := Q2
    end;
2: begin
    choicevector[3] := Q1;
    choicevector[4] := Q2
    end;
3: begin
    choicevector[5] := Q1;
    choicevector[6] := Q2
    end;
4: begin
    choicevector[7] := Q1;
    choicevector[8] := Q2
    end;
5: begin
    choicevector[9] := Q1;
    choicevector[10] := Q2
    end;
6: begin
    choicevector[11] := Q1;
    choicevector[12] := Q2
    end;
7: begin
    choicevector[13] := Q1;
    choicevector[14] := Q2
    end;
8: begin
    choicevector[15] := Q1;
    choicevector[16] := Q2
    end;

```



```

          9: begin
              choicevector[17] := Q1;
              choicevector[18] := Q2
          end;
      10: begin
          choicevector[19] := Q1;
          choicevector[20] := Q2
      end
      end; (case of counter)
      counter := counter + 1
  end      (for j loop)
end      (for i loop)
end; (OptionComparisons)

```

```

overlay procedure PrintOptionComparisons(choicevector:
                                         outputdatafile; numoptions: integer);

```

```

(PrintOptionComparisons - Prints the comparisons made by a user.
Prints the number of options determined by numoptions.
Sends the print to the computers list device.)

```

```

label 1,3,4,5;

```

```

begin
  write(Lst,b,'                               Scale                               ');
  writeln(Lst,'      Scale');
  write(Lst,b,'      Comparison      Value      Comparison');
  writeln(Lst,'      Value');
  write(Lst,b,'      -----      -----      -----');
  writeln(Lst,'      -----');
  case numoptions of
    2: ;
    3: GoTo 3;
    4: GoTo 4;
    5: GoTo 5
  end; (case of numoptions)
  (*****)
  if choicevector[1] = '1' then write(Lst,b,'Option 1 over 2')
  else write(Lst,b,'Option 2 over 1');
  writeln(Lst,'      ',choicevector[2]);
  GoTo 1;
  (*****)
  3:if choicevector[1] = '1' then write(Lst,b,'Option 1 over 2')
  else write(Lst,b,'Option 2 over 1');
  write(Lst,'      ',choicevector[2],'      ');
  if choicevector[3] = '1' then write(Lst,'1 over 3      ')
  else write(Lst,'3 over 1      ');
  writeln(Lst,choicevector[4]);
  if choicevector[5] = '2' then write(Lst,b,'Option 2 over 3')
  else write(Lst,b,'Option 3 over 2');
  writeln(Lst,'      ',choicevector[6]);
  GoTo 1;
  (*****)

```

```

4:if choicevector[1] = '1' then write(Lst,b,'Option 1 over 2')
   else write(Lst,b,'Option 2 over 1');
   write(Lst,' ',choicevector[2],' ');
   if choicevector[3] = '1' then write(Lst,'1 over 3 ');
   else write(Lst,'3 over 1 ');
   writeln(Lst,choicevector[4]);
   if choicevector[5] = '1' then write(Lst,b,'Option 1 over 4')
   else write(Lst,b,'Option 4 over 1');
   write(Lst,' ',choicevector[6],' ');
   if choicevector[7] = '2' then write(Lst,'2 over 3 ');
   else write(Lst,'3 over 2 ');
   writeln(Lst,choicevector[8]);
   if choicevector[9] = '2' then write(Lst,b,'Option 2 over 4')
   else write(Lst,b,'Option 4 over 2');
   write(Lst,' ',choicevector[10],' ');
   if choicevector[11] = '3' then write(Lst,'3 over 4 ');
   else write(Lst,'4 over 3 ');
   writeln(Lst,choicevector[12]);
   GoTo 1;
(*****)
5:if choicevector[1] = '1' then write(Lst,b,'Option 1 over 2')
   else write(Lst,b,'Option 2 over 1');
   write(Lst,' ',choicevector[2],' ');
   if choicevector[3] = '1' then write(Lst,'1 over 3 ');
   else write(Lst,'3 over 1 ');
   writeln(Lst,choicevector[4]);
   if choicevector[5] = '1' then write(Lst,b,'Option 1 over 4')
   else write(Lst,b,'Option 4 over 1');
   write(Lst,' ',choicevector[6],' ');
   if choicevector[7] = '1' then write(Lst,'1 over 5 ');
   else write(Lst,'5 over 1 ');
   writeln(Lst,choicevector[8]);
   if choicevector[9] = '2' then write(Lst,b,'Option 2 over 3')
   else write(Lst,b,'Option 3 over 2');
   write(Lst,' ',choicevector[10],' ');
   if choicevector[11] = '2' then write(Lst,'2 over 4 ');
   else write(Lst,'4 over 2 ');
   writeln(Lst,choicevector[12]);
   if choicevector[13] = '2' then write(Lst,b,'Option 2 over 5')
   else write(Lst,b,'Option 5 over 2');
   write(Lst,' ',choicevector[14],' ');
   if choicevector[15] = '3' then write(Lst,'3 over 4 ');
   else write(Lst,'4 over 3 ');
   writeln(Lst,choicevector[16]);
   if choicevector[17] = '3' then write(Lst,b,'Option 3 over 5')
   else write(Lst,b,'Option 5 over 3');
   write(Lst,' ',choicevector[18],' ');
   if choicevector[19] = '4' then write(Lst,'4 over 5 ');
   else write(Lst,'5 over 4 ');
   writeln(Lst,choicevector[20]);
(*****)
1:writeln(Lst,' ')
end; (PrintOptionComparisons)

```

```

overlay procedure WriteOptionComparisons(choicevector:
                                     outputdatafile; numoptions: integer);

(WriteOptionComparisons - Same as PrintOptionComparisons, except
 this procedure prints to the screen.)

label 1,3,4,5;

begin
  write('                               Scale                               ');
  writeln('      Scale');
  write('      Comparison      Value      Comparison');
  writeln('      Value');
  write('      -----      -----      -----');
  writeln('      -----');
  case numoptions of
    2: ;
    3: GoTo 3;
    4: GoTo 4;
    5: GoTo 5
  end; (case of numoptions)
  (*****
    if choicevector[1] = '1' then write('Option 1 over 2')
    else write('Option 2 over 1');
    writeln('      ',choicevector[2]);
    GoTo 1;
  (*****
    3:if choicevector[1] = '1' then write('Option 1 over 2')
    else write('Option 2 over 1');
    write('      ',choicevector[2],'      ');
    if choicevector[3] = '1' then write('1 over 3      ')
    else write('3 over 1      ');
    writeln(choicevector[4]);
    if choicevector[5] = '2' then write('Option 2 over 3')
    else write('Option 3 over 2');
    writeln('      ',choicevector[6]);
    GoTo 1;
  (*****
    4:if choicevector[1] = '1' then write('Option 1 over 2')
    else write('Option 2 over 1');
    write('      ',choicevector[2],'      ');
    if choicevector[3] = '1' then write('1 over 3      ')
    else write('3 over 1      ');
    writeln(choicevector[4]);
    if choicevector[5] = '1' then write('Option 1 over 4')
    else write('Option 4 over 1');
    write('      ',choicevector[6],'      ');
    if choicevector[7] = '2' then write('2 over 3      ')
    else write('3 over 2      ');
    writeln(choicevector[8]);
    if choicevector[9] = '2' then write('Option 2 over 4')
    else write('Option 4 over 2');
    write('      ',choicevector[10],'      ');
  )
  )
  )

```

```

        if choicevector[11] = '3' then write('3 over 4      ');
        else write('4 over 3      ');
        writeln(choicevector[12]);
        GoTo 1;
    (*****)
    5:if choicevector[1] = '1' then write('Option 1 over 2')
        else write('Option 2 over 1');
        write('      ',choicevector[2],');
        if choicevector[3] = '1' then write('1 over 3      ');
        else write('3 over 1      ');
        writeln(choicevector[4]);
        if choicevector[5] = '1' then write('Option 1 over 4')
            else write('Option 4 over 1');
        write('      ',choicevector[6],');
        if choicevector[7] = '1' then write('1 over 5      ');
        else write('5 over 1      ');
        writeln(choicevector[8]);
        if choicevector[9] = '2' then write('Option 2 over 3')
            else write('Option 3 over 2');
        write('      ',choicevector[10],');
        if choicevector[11] = '2' then write('2 over 4      ');
        else write('4 over 2      ');
        writeln(choicevector[12]);
        if choicevector[13] = '2' then write('Option 2 over 5')
            else write('Option 5 over 2');
        write('      ',choicevector[14],');
        if choicevector[15] = '3' then write('3 over 4      ');
        else write('4 over 3      ');
        writeln(choicevector[16]);
        if choicevector[17] = '3' then write('Option 3 over 5')
            else write('Option 5 over 3');
        write('      ',choicevector[18],');
        if choicevector[19] = '4' then write('4 over 5      ');
        else write('5 over 4      ');
        writeln(choicevector[20]);
    (*****)
    1:writeln
    end; (WriteOptionComparisons)

```

```
(*****
*****      Include file: extra2.dss      *****
*****)
```

```
procedure ChangeOptionInformation(Var optionrec: conceptoption);
```

(ChangeOptionInformation - this procedure is part of the database management system and is only called by the procedure DataBase Management. It accepts a record of concept option (all the information concerning a concept option and provides an opportunity to change that information.

Variables - i,j: counting integers.

numti: integer counter of the number of technology issues for this concept.

answer: character to receive input.

OK: boolean to determine if input is real when it must be, integer when it must be, etc.)

```
label 2,3,4;
```

```
Var i,j,numti: integer;
    tempoption: conceptoption;
    answer: char;
    OK: boolean;
```

```
begin
```

```
    InitializeConceptOption(tempoption);
```

```
    tempoption := optionrec;
```

```
2:ClrScr;
```

```
    writeln;
```

```
    writeln(b,'      Select Appropriate Number');
```

```
    writeln;
```

```
    writeln(b,'      CHANGE OPTIONS');
```

```
    writeln(b,'      -----');
```

```
(ChangeOption)
```

```
    writeln(b,'      1) Option Name');
```

```
(Menu)
```

```
    writeln(b,'      2) Option Originator');
```

```
    writeln(b,'      3) Technology Issue');
```

```
    writeln(b,'      4) Performance Value');
```

```
    writeln(b,'      5) Return to last menu');
```

```
    writeln;
```

```
    write(b,'      SELECTION: ');
```

```
    readln(answer);
```

```
    while not (answer in ['1'..'5']) do
```

```
    begin
```

```
        writeln;
```

```
        write(b,'Incorrect selection, not (1..5), RESELECT: ');
```

```
        readln(answer)
```

```
    end;
```

```
    with tempoption do
```

```
    begin
```

```
        case answer of
```

```
            '1': begin
```

```

answer := 'n';
while answer in ['N','n'] do
begin
  ClrScr;
  writeln;
  writeln;
  write(b,'Current option name is ');
  write(optionname,', change to (20 char): ');
  readln(optionname);
  writeln;
  write(b,'Is ',optionname,' correct ');
  write('(Y or N)? ');
  readln(answer);
  CheckYorN(answer)
end; (while answer in no)
GoTo 2
end; (case of 1 - change option name)
'2': begin
  answer := 'n';
  while answer in ['N','n'] do
  begin
    ClrScr;
    writeln;
    writeln;
    write(b,'Current option originator is ');
    write(optionorigin,', change to (20 char): ');
    readln(optionorigin);
    writeln;
    write(b,'Is ',optionorigin,' correct ');
    write('(Y or N)? ');
    readln(answer);
    CheckYorN(answer)
  end; (while answer in no)
  GoTo 2;
end; (case of 1 - change option name)
'3': begin
  3:ClrScr;
  OK := false;
  writeln;
  write(b,'Select Technology Issue to change ');
  writeln('by its #');
  for i := 1 to 15 do
    if optionti[i].tiname = '' then
      numti := numti + 1;
  for i := 1 to numti do
    writeln(' ',i:2,') ',optionti[i].tiname);
  writeln(' ',i+1:2,') Change none ');
  writeln;
  write(b,b,'SELECTION: ');
  while not OK do
  begin
    ($I-) (Shuts of automatic error checking)
    readln(j);

```

```

($I+)      (Turns on automatic error checking)
OK := (IOresult = 0);
if not OK or not (j in [1..i+1]) then
begin
    write(b,b,'Not an option, Reselect: ');
    OK := false
end
end; (while not OK)
if j = i + 1 then GoTo 2;
ClrScr;
writeln;
writeln(b,optionti[j].tiname);
write(b,'Change Tech Issue Name (Y or N)? ');
readln(answer);
CheckYorN(answer);
while answer in ['Y','y'] do
begin
    write('Change to (up to 80 char): ');
    readln(optionti[j].tiname);
    writeln('Is ',optionti[j].tiname);
    write('correct (Y or N)? ');
    readln(answer);
    CheckYorN(answer);
    if answer in ['N','n'] then answer := 'y'
    else answer := 'n'
end; (while changing tiname)
write(b,'Schedule is ',optionti[j].tischedule);
write(' ', do you wish to change (Y or N)? ');
readln(answer);
while answer in ['Y','y'] do
begin
    OK := false;
    write(b,'Change to: ');
    while not OK do
begin
($I-)      (Shuts of automatic error checking)
    readln(optionti[j].tischedule);
($I+)      (Turns on automatic error checking)
    OK := (IOresult = 0);
    if not OK then
begin
        write(b,b,'Enter # again: ');
        OK := false
    end
end; (while not OK)
    write(b,'Is ',optionti[j].tischedule);
    write(' ', correct (Y or N)? ');
    readln(answer);
    if answer in ['N','n'] then answer := 'y'
    else answer := 'n'
end; (while changing tischedule)
write(b,'Risk is ',optionti[j].tirisk);
write(' ', do you wish to change (Y or N)? ');

```

```

readln(answer);
while answer in ['Y','y'] do
begin
  OK := false;
  write(b,'Change to: ');
  while not OK do
  begin
    (SI-) (Shuts of automatic error checking)
    readln(optionti[j].tirisk);
    (SI+) (Turns on automatic error checking)
    OK := (IOresult = 0);
    if not OK then
    begin
      write(b,b,'Enter # again: ');
      OK := false
    end
  end; (while not OK)
  write(b,'Is ',optionti[j].tirisk);
  write(' ', correct (Y or N)? ');
  readln(answer);
  if answer in ['N','n'] then answer := 'y'
  else answer := 'n'
  end; (while changing tirisk)
write(b,'Cost is ',optionti[j].ticost);
write(' ', do you wish to change (Y or N)? ');
readln(answer);
while answer in ['Y','y'] do
begin
  OK := false;
  write(b,'Change to: ');
  while not OK do
  begin
    (SI-) (Shuts of automatic error checking)
    readln(optionti[j].ticost);
    (SI+) (Turns on automatic error checking)
    OK := (IOresult = 0);
    if not OK then
    begin
      write(b,b,'Enter # again: ');
      OK := false
    end
  end; (while not OK)
  write(b,'Is ',optionti[j].ticost);
  write(' ', correct (Y or N)? ');
  readln(answer);
  if answer in ['N','n'] then answer := 'y'
  else answer := 'n'
  end; (while changing ticost)
write(b,'Would you like to change another ');
write('Technology Issue (Y or N)? ');
readln(answer);
if answer in ['Y','y'] then GoTo 3;
GoTo 2

```



```

        end; (case of change a Tech Issue)
'4': begin
    with optionperformance do
    begin
        while answer in ['Y','y'] do
        begin
            ClrScr;
            writeln;
            writeln(b,'Select # of Value to Change');
            writeln;
            writeln(b,'    Performance Values');
            writeln(b,'    -----');
            writeln(b,' 1) Coverage    = ',coverage);
            writeln(b,' 2) Capacity    = ',capacity);
            writeln(b,' 3) Quality      = ',quality);
            writeln(b,' 4) Timeliness = ',timeliness);
            write(b,' 5) Availability = ');
            writeln(availability);
            write(b,' 6) Survivability = ');
            writeln(survivability);
            writeln(b,' 7) None');
            writeln;
            write(b,'    SELECTION: ');
            readln(answer);
            while not (answer in ['1'..'7']) do
            begin
                write(b,'    Reselect: ');
                readln(answer)
            end;
            if answer = '7' then answer := 'n';
            if answer in ['1'..'6'] then
            begin
                OK := false;
                write(b,'Change to: ');
                while not OK do
                begin
(SI-)    (Shuts of automatic error checking)
                    readln(i);
(SI+)    (Turns on automatic error checking)
                    OK := (IOresult = 0);
                    if not OK then
                    begin
                        write(b,b,'Enter # again: ');
                        OK := false
                    end
                end; (while not OK)
                case answer of
                    '1': coverage := i;
                    '2': capacity := i;
                    '3': quality := i;
                    '4': timeliness := i;
                    '5': availability := i;
                    '6': survivability := i

```

```

        end; (case of answer)
        answer := 'y';
        end (if answer in 1 to 6)
        end (while answer is yes)
        end; (with performance do)
        GoTo 2
        end; (case answer of '4')
    '5':
    end (case statement)

    end; (with tempoption do)
    optionrec := tempoption
    end; (ChangeOptionInformation)

overlay procedure Quit (Var quitindicator:integer);

(Quit - determines if a user really wants to quit. Uses the global
variable quitindicator. If user wants to quit, Quit returns
a value of 1. If not, a value of 0.
Object code for this procedure will be in dss.001.

Calls CheckYorN

Variable - query: character to receive input.
           i: counting integer.)

Var query: char;
    i: integer;

begin
    i := 0;
    ClrScr;
    for i := 1 to 10 do writeln;
    writeln(b,'Do you wish to leave the program now?');
    write(b,'      Yes or No ("Y" or "N")? ');
    readln(query);
    CheckYorN(query);
    writeln;
    case query of
        'Y','y':begin
            ClrScr;
            for i := 1 to 10 do writeln;
            write(b,'You ');
            writeln('have terminated execution of the');
            write(b,'AFSTC ');
            writeln('Decision Support System (DSS)');
            write(b);
            writeln('for Space System Tradeoffs. ');
            quitindicator := 1
            end;
        'N','n':begin
            ClrScr;
            for i := 1 to 10 do writeln;

```

```

        write(b, '    Wait, you ');
        writeln('will be returned to the program');
        Delay(1500);
        quitindicator := 0;
    end
end (case query of Y or N)
end; (Quit)

```

overlay procedure InputNewConcept(Var newconceptrecord: datafile);

(InputNewConcept - takes a record of type datafile and receives a new concept name and originator.

Calls CheckYorN

Variables - i,j: counting integers.
 answer: a character to receive input.)

label 2,3;

Var i,j: integer;
 answer: char;

```

begin
    ClrScr;
    with newconceptrecord do
        begin
            writeln;
            write('    The new concept name, and the concepts ');
            writeln('originator,');
            write('    both no more than 20 characters, will ');
            writeln('be input at');
            writeln('    this time.');
```

2:writeln;

```

            write('    Enter the new concept name: ');
            readln(conceptname);
            writeln;
            write('    Is ',conceptname,' correct (Y or N)? ');
            readln(answer);
            CheckYorN(answer);
            if answer in ['N','n'] then goto 2;
            writeln;
```

3:writeln;

```

            write('    Enter the concept originator: ');
            readln(conceptorigin);
            writeln;
            write('    Is ',conceptorigin,' correct? (Y or N)? ');
            readln(answer);
            CheckYorN(answer);
            if answer in ['N','n'] then goto 3
        end (with newconceptrecord)
    end;
    (InputNewConcept)
end;

```

```

overlay procedure InputConceptOptions(filename: Char12;
                                     answer: char);

```

(InputConceptOptions - Governing procedure that determines what operation will be performed on a concept record. Receives a filename of the appropriate file in the database and a character answer.

Calls: InitializeConceptOption; ChangeOptionInformation; CheckYorN.

Variables - optionfile, a file of conceptoption records.
 optionrec: a record of concept options.
 i,j: integer counting variable.
 numoptions,numissues: # of options and tech issues res.
 option,origin: strings of up to 20 characters.
 tempanswer: a character holding value.
 OK: boolean: to determine if input is correct format.)

label 1,2,3,4;

```

Var  optionfile: file of conceptoption;
     optionrec: conceptoption;
     i,j,numoptions,numissues: integer;
     option, origin: Char20;
     tempanswer: char;
     OK: boolean;

```

```

begin
  Assign(optionfile,filename);
  InitializeConceptOption(optionrec);
  if answer = '1' then
    begin
      Rewrite(optionfile);
      GoTo 3
    end;
  Reset(optionfile);
2:ClrScr;
  writeln;
  writeln(b,'    Choose the appropriate number');
  writeln;
  writeln(b,'          Input Options');
  writeln(b,'          -----');
  writeln(b,'          1) ADD an option');
  writeln(b,'          2) CHANGE an option');
  writeln(b,'          [or part of one]');
  writeln(b,'          3) DELETE an option');
  writeln(b,'          4) None of the above');
  writeln;
  write(b,'          SELECTION: ');
  readln(answer);
  while not (answer in ['1','2','3','4']) do
    begin
      write(b,answer,' not an option.  Reselect: ');

```

```

    readln(answer)
end;
case answer of
  '1': begin
    if FileSize(optionfile) = maxnumoptions then
      begin
        writeln;
        write(b,'Only ',maxnumoptions,' options allowed ');
        writeln('per concept. You must remove');
        write(b,'an option before you can input ');
        writeln('another. ');
        Delay(1500);
        GoTo 2
      end; (if over maxnumconcepts)
      Seek(optionfile,FileSize(optionfile));
      answer := 'y';
      GoTo 3
    end; (answer of '1')
  '2': begin
    numoptions := 0;
    Seek(optionfile,0);
    Read(optionfile,optionrec);
    while (optionrec.optionname <> '')
      do numoptions := numoptions + 1;

    ClrScr;
    Seek(optionfile,0);
    writeln;
    writeln;
    writeln(b,'Select number of option to work with');
    writeln;
    writeln(b,'          Concept Options');
    writeln(b,'          -----');
    for i := 1 to numoptions do
      begin
        Read(optionfile,optionrec);
        writeln(b,'          ',i,' ',optionrec.optionname)
      end;
    writeln;
    write(b,'          SELECTION: ');
    readln(answer);
    while not (answer in ['1'..chr(numoptions+48)]) do
      begin
        write(b,answer,' not an option, ');
        write('please reselect: ');
        readln(answer)
      end;
    Seek(optionfile,ord(answer)-49);
    Read(optionfile,optionrec);
    ChangeOptionInformation(optionrec);
    Seek(optionfile,ord(answer)-49);
    Write(optionfile,optionrec);
    GoTo 2
  end; (case of Changing an Option)

```

```

'3': begin
    numoptions := 0;
    Seek(optionfile,0);
    Read(optionfile,optionrec);
    while (optionrec.optionname <> '')
        do numoptions := numoptions + 1;

    ClrScr;
    Seek(optionfile,0);
    writeln;
    writeln;
    writeln(b,'Select number of option to DELETE');
    writeln;
    writeln(b,'          Concept Options');
    writeln(b,'          -----');
    for i := 1 to numoptions do
        begin
            Read(optionfile,optionrec);
            writeln(b,'          ',i,'')',optionrec.optionname)
        end;
    writeln(b,'          ',i+1,'')', 'None');
    writeln;
    write(b,'          SELECTION: ');
    readln(answer);
    while not (answer in ['1'..chr(numoptions+49)]) do
        begin
            write(b,answer,' not an option, please reselect: ');
            readln(answer)
        end;
    if answer = chr(numoptions + 49) then GoTo 2;
    tempanswer := answer;
    Seek(optionfile,ord(answer)-49);
    writeln;
    write(b,'Confirm removal of ',optionrec.optionname);
    write(' (Y or N)? ');
    readln(answer);
    if answer in ['n','N'] then GoTo 2;
    if (ord(tempanswer)-48) <> numoptions then
        begin
            Seek(optionfile,numoptions-1);
            Read(optionfile,optionrec);
            Seek(optionfile,ord(tempanswer)-49);
            Write(optionfile,optionrec)
        end; (if)
    InitializeConceptOption(optionrec);
    Seek(optionfile,numoptions-1);
    Write(optionfile,optionrec);
    GoTo 2
end; (if answer is to delete)
'4': GoTo 1
end; (case of answer for main menu)
3:ClrScr;
writeln;
with optionrec do

```

```

begin
  write(b,'What is the new option name? ');
  readln(optionname);
  write(b,'Is ',optionname,' correct (Y or N)? ');
  readln(answer);
  if answer in ['N','n'] then
    begin
      write(b,'Change, ',optionname:20,' to: ');
      readln(optionname)
    end;
  write(b,'Who is the option originator: ');
  readln(optionorigin);
  write(b,'Is ',optionorigin,' correct (Y or N)? ');
  readln(answer);
  CheckYorN(answer);
  if answer in ['N','n'] then
    begin
      write(b,'Change, ',optionorigin,' to: ');
      readln(optionorigin)
    end;
  writeln;
  write('      How many Tech Issues do you wish to ');
  write('input (integer: 1-15)? ');
  repeat
    ($I-)
    readln(numissues);
    ($I+)
    OK := (IOresult = 0);
    if not OK then write('      Must be integer(1-15)');
  until OK;
  while not (numissues in [1..15]) do
    begin
      write(b,'This data base only allows 15 technology ');
      writeln('issues per concept option. ');
      write(b,'Please reselect: ');
      readln(numissues)
    end; {while numissues not in 1-15}
  for j := 1 to numissues do
    begin
      ClrScr;
      writeln;
      write('      Technology Issue - ',j,' out of ');
      writeln(numissues);
      writeln;
      write('      What is the Tech Issue name? ');
      readln(optionti[j].tiname);
      write('      What is the Tech Issue risk ');
      write('(1 to 5)? ');
      readln(optionti[j].tirisk);
      write('      What is scheduled number of years ');
      write('to completion? ');
      readln(optionti[j].tischedule);
      write('      What is the cost to solve the issue? ');
    end;
end;

```

```

readln(optionti[j].ticost);
writeln;
writeln('      Name: ',optionti[j].tiname);
write('      Schedule: ',optionti[j].tischedule:2);
write('      Risk: ',optionti[j].tirisk,'      Cost: ');
writeln(optionti[j].ticost:8);
write('      Is the above correct (Y or N)? ');
readln(answer);
CheckYorN(answer);
if answer in ['N','n'] then j := j - 1
end; (for j 1 to numissues)
if numissues < 15 then
begin
for j := (numissues+1) to 15 do
begin
optionti[j].tiname := '';
optionti[j].tirisk := 0;
optionti[j].tischedule := 0;
optionti[j].ticost := 0.0
end (for j)
end; (if numissues < 15)
with optionperformance do
begin
4:ClrScr;
writeln;
writeln(b,'      ',optionname);
writeln(b,'Performance values (1 - 5)');
writeln;
write(b,'What is the coverage value? ');
readln(coverage);
write(b,'What is the capacity value? ');
readln(capacity);
write(b,'What is the quality value? ');
readln(quality);
write(b,'What is the timeliness value? ');
readln(timeliness);
write(b,'What is the availability value? ');
readln(availability);
write(b,'What is the survivability value? ');
readln(survivability);
writeln;
writeln(b,'Cov Cap Qua Tim Ava Sur');
write(b,' ',coverage,' ',capacity,' ',quality);
write(' ',timeliness,' ',availability,' ');
writeln(survivability);
write('      Are the above performance values ');
write('correct (Y or N)? ');
readln(answer);
CheckYorN(answer);
if answer in ['N','n'] then GoTo 4
end (with optionperformance)
end; (with optionrec)
Write(optionfile,optionrec);

```



```

    GoTo 2;
1:ClrScr;
   for i := 1 to 11 do writeln;
   writeln(b,'Input Program Terminated');
   Close(optionfile)
end;      (InputConceptOption)

overlay procedure PrintPreviousRun(Var userdata: userfile;
                                   section: integer);

(PrintPreviousRun - prints to printer a previous user run represented
by the userdata variable. The section to be printed is
the variable section.
Object code stored in dss.001.

Calls - PrintOptionComparisons

Variables i,numoptions: counting integers.)

label 1,2,3,4,5,6;

Var i,numoptions: integer;

begin
  numoptions := 0;
  for i := 1 to maxnumoptions do
    if (userdata.options[i] <> ' ') then
      numoptions := numoptions + 1;

  with userdata do
    begin
      case section of
        1: ;
        2: GoTo 2;
        3: GoTo 3;
        4: GoTo 4;
        5: GoTo 5;
        6: GoTo 6
      end; (case of section)
      ClrScr;
      write(Lst,b,' ');
      writeln(Lst,'PRIORITIZATION OF ATTRIBUTES (CRITERIA)');
      writeln(Lst,' ');
      writeln(Lst,b,' ',conceptname);
      write(Lst,b,' Criteria Priority');
      writeln(Lst,'-Weight');
      write(Lst,b,' -----');
      writeln(Lst,'-----');
      writeln(Lst,' ');
      write(Lst,b,' Performance ');
      with priorityvectors do
        begin
          writeln(Lst,criteriapriorities[11:8];

```

```

write(Lst,b,'          Schedule          ');
writeln(Lst,criteriapriorities[2]:8);
write(Lst,b,'          Risk          ');
writeln(Lst,criteriapriorities[3]:8);
write(Lst,b,'          Cost          ');
writeln(Lst,criteriapriorities[4]:8);
writeln(Lst,' ');
end; (with priorityvectors do)
writeln(Lst,b,'          Consistency Ratio is: ',CRvector[1]:6);
writeln(Lst,' ');
writeln(Lst,b,'          Comparison          Scale Value');
writeln(Lst,b,'          -----          -----');
with judgements do
begin
  if criteriachoices[1] in ['P','p'] then
    write(Lst,b,'          Performance over Schedule ')
  else
    write(Lst,b,'          Schedule over Performance ');
  writeln(Lst,'          ',criteriachoices[2]);

  if criteriachoices[3] in ['P','p'] then
    write(Lst,b,'          Performance over Risk ')

  else
    write(Lst,b,'          Risk over Performance ');
  writeln(Lst,'          ',criteriachoices[4]);
  if criteriachoices[5] in ['P','p'] then
    write(Lst,b,'          Performance over Cost ')
  else
    write(Lst,b,'          Cost over Performance ');
  writeln(Lst,'          ',criteriachoices[6]);
  if criteriachoices[7] in ['S','s'] then
    write(Lst,b,'          Schedule over Risk ')
  else
    write(Lst,b,'          Risk over Schedule ');
  writeln(Lst,'          ',criteriachoices[8]);
  if criteriachoices[9] in ['S','s'] then
    write(Lst,b,'          Schedule over Cost ')
  else
    write(Lst,b,'          Cost over Schedule ');
  writeln(Lst,'          ',criteriachoices[10]);
  if criteriachoices[11] in ['R','r'] then
    write(Lst,b,'          Risk over Cost ')
  else
    write(Lst,b,'          Cost over Risk ');
  writeln(Lst,'          ',criteriachoices[12])
end; (with judgements do)
writeln(Lst,' ');
GoTo 1;
(*****)
2:ClrScr;
writeln(Lst,b,'          ATTRIBUTE - PERFORMANCE');
writeln(Lst,b,'          ',conceptname);

```

```

writeln(Lst,b,' ');
write(Lst,b,'          Option Name          ');
writeln(Lst,'Priority');
write(Lst,b,'          -----          ');
writeln(Lst,'-----');
writeln(Lst,b,' ');
for i := 1 to numoptions do
begin
    write(Lst,b,' ',i,' ') ',options[i]:20,' ');
    writeln(Lst,priorityvectors.performancevector[i]:8)
end;
writeln(Lst,' ');
writeln(Lst,b,'          Consistency Ratio: ',CRvector[2]:6);
writeln(Lst,b,' ');
write(Lst,'          Based on the criteria of performance');
writeln(Lst,' your comparisons were:');
PrintOptionComparisons(judgements.performancechoices,
                        numoptions);

GoTo 1;
(*****)
3:ClrScr;
writeln(Lst,b,'          ATTRIBUTE - SCHEDULE');
writeln(Lst,b,'          ',conceptname);
writeln(Lst,b,' ');
writeln(Lst,b,'          Option Name          Priority');
writeln(Lst,b,'          -----          -----');
writeln(Lst,b,' ');
for i := 1 to numoptions do
begin
    write(Lst,b,' ',i,' ') ',options[i]:20,' ');
    writeln(Lst,priorityvectors.schedulevector[i]:8)
end;
writeln(Lst,b,' ');
writeln(Lst,b,'          Consistency Ratio: ',CRvector[3]:6);
writeln(Lst,b,' ');
write(Lst,'          Based on the criteria of schedule, ');
writeln(Lst,' your comparisons were:');
PrintOptionComparisons(judgements.schedulechoices,
                        numoptions);

GoTo 1;
(*****)
4:ClrScr;
writeln(Lst,b,'          ATTRIBUTE - RISK');
writeln(Lst,b,'          ',conceptname);
writeln(Lst,b,'          Option          Priority');
writeln(Lst,b,'          -----          -----');
writeln(Lst,b,' ');
for i := 1 to numoptions do
begin
    write(Lst,b,' ',i,' ') ',options[i]:20';
    writeln(Lst,'          ',priorityvectors.riskvector[i]:8);
end;
writeln(Lst,b,' ');

```

```

writeln(Lst,b,'      Consistency Ratio: ',CRvector[4]:6);
writeln(Lst,b,' ');
write(Lst,'      Based on the criteria of risk, your ');
writeln(Lst,'comparisons were:');
PrintOptionComparisons(judgements.riskchoices,numoptions);
GoTo 1;
(*****)
5:ClrScr;
writeln(Lst,b,'      ATTRIBUTE - COST');
writeln(Lst,b,'      ',conceptname);
writeln(Lst,b,'      Option      Priority');
writeln(Lst,b,'      -----      -----');
writeln(Lst,b,' ');
for i := 1 to numoptions do
begin
write(Lst,b,'      ',i,' ') options[i]:20;
writeln(Lst,b,'      ',priorityvectors.costvector[i]:8);
end;
writeln(Lst,b,' ');
writeln(Lst,b,'      Consistency Ratio: ',CRvector[5]:6);
writeln(Lst,b,' ');
write(Lst,b,'      Based on the criteria of cost, your ');
writeln(Lst,'comparisons were:');
PrintOptionComparisons(judgements.costchoices,numoptions);
GoTo 1;
(*****)
6:ClrScr;
writeln(Lst,b,'      FINAL PRIORITIES');
writeln(Lst,b,'      -----');
writeln(Lst,b,'      ',conceptname);
writeln(Lst,b,'      Option      Priority');
writeln(Lst,b,'      -----      -----');
for i := 1 to numoptions do
begin
write(Lst,b,'      ',i,' ') options[i]:20,'      ');
writeln(Lst,priorityvectors.finalpriorities[i]:8);
end;
writeln(Lst,b,' ');
write(Lst,'      This is the final priority of the ');
writeln(Lst,'options for this concept,');
write(Lst,'      based upon your judgements of the ');
writeln(Lst,'importance of the criteria');
write(Lst,'      and the ',numoptions,' options com');
writeln(Lst,'paired against each other, with respect');
writeln(Lst,'      to the criteria. ');
writeln(Lst,' ');
(*****)
end; (with userdata do)
1:writeln
end; (PrintPreviousRecord)

```

```

overlay procedure ViewPreviousRun(Var userdata: userfile;
                                section: integer);
{ViewPreviousRun - Same as PrintPreviousRun except this procedure
 prints to the screen.
 Object code is assigned to dss.001.

 Call view option comparisons.

 Variables(extra) : hold, a character to hold the screen for a
 return.)

label 1,2,3,4,5,6;

Var i,j,numoptions: integer;
    hold: char;

begin
    numoptions := 0;
    for i := 1 to maxnumoptions do
        if (userdata.options[i] <> ' ') then
            numoptions := numoptions + 1;
    with userdata do
        begin
            case section of
                1: ;
                2: GoTo 2;
                3: GoTo 3;
                4: GoTo 4;
                5: GoTo 5;
                6: GoTo 6
            end; {case of section}
            ClrScr;
            writeln('          PRIORITIZATION OF CRITERIA (ATTRIBUTES)');
            writeln('          ',conceptname);
            writeln('          Criteria          Priority-Weight');
            writeln('          -----          -----');
            writeln;
            write('          Performance          ');
            with priorityvectors do
                begin
                    writeln(criteriapriorities[1]:8);
                    write('          Schedule          ');
                    writeln(criteriapriorities[2]:8);
                    write('          Risk          ');
                    writeln(criteriapriorities[3]:8);
                    write('          Cost          ');
                    writeln(criteriapriorities[4]:8);
                end; {with priorityvectors do}
            writeln;
            writeln('          Consistency Ratio is: ',CRvector[1]:6);
            writeln;
            writeln('          Comparison          Scale Value');

```

```

writeln('-----');
with judgements do
begin
  if criteriachoices[1] in ['P','p'] then
    write('      Performance over Schedule      ')
  else
    write('      Schedule over Performance      ');
writeln(criteriachoices[2]);
  if criteriachoices[3] in ['P','p'] then
    write('      Performance over Risk      ')
  else
    write('      Risk over Performance      ');
writeln(criteriachoices[4]);
  if criteriachoices[5] in ['P','p'] then
    write('      Performance over Cost      ')
  else
    write('      Cost over Performance      ');
writeln(criteriachoices[6]);
  if criteriachoices[7] in ['S','s'] then
    write('      Schedule over Risk      ')
  else
    write('      Risk over Schedule      ');
writeln(criteriachoices[8]);
  if criteriachoices[9] in ['S','s'] then
    write('      Schedule over Cost      ')
  else
    write('      Cost over Schedule      ');
writeln(criteriachoices[10]);
  if criteriachoices[11] in ['R','r'] then
    write('      Risk over Cost      ')
  else
    write('      Cost over Risk      ');

  writeln(criteriachoices[12])
end; (with judgements do)
writeln;
write('      Press RETURN when finished reading: ');
readln(hold);
GoTo 1;
(*****)
2:ClrScr;
writeln(b,'      ATTRIBUTE - PERFORMANCE');
writeln(b,'      ',conceptname);
writeln;
writeln(b,'      Option Name      Priority');
writeln(b,'      -----      -----');
writeln;
for i := 1 to numoptions do
begin
  write(b,i,' ') ',options[i]:20,' ');
  writeln(priorityvectors.performancevector[i]:8)
end;
writeln;

```

```

writeln(b,'          Consistency Ratio: ',CRvector[2]:6);
writeln;
write('          Based on the criteria of performance, your ');
writeln('comparisons were:');
WriteOptionComparisons(judgements.performancechoices,
                        numoptions);

write('          To continue, press RETURN: ');
readln(hold);
GoTo 1;
(*****)
3:ClrScr;
writeln(b,'          ATTRIBUTE - SCHEDULE');
writeln(b,'          ',conceptname);
writeln;
writeln(b,'          Option Name          Priority');
writeln(b,'          -----          -----');
writeln;
for i := 1 to numoptions do
begin
write(b,i,' ') ',options[i]:20,' ');
writeln(priorityvectors.schedulevector[i]:8)
end;
writeln;
writeln(b,'          Consistency Ratio: ',CRvector[3]:6);
writeln;
write('          Based on the criteria of schedule, your ');
writeln('comparisons were:');
WriteOptionComparisons(judgements.schedulechoices,
                        numoptions);

write(b,'To continue, press RETURN: ');
readln(hold);
GoTo 1;
(*****)
4:ClrScr;
writeln(b,'          ATTRIBUTE - RISK');
writeln(b,'          ',conceptname);
writeln(b,'          Option          Priority');
writeln(b,'          -----          -----');
writeln;
for i := 1 to numoptions do
begin
write('          ',i,' ') ',options[i]:20);
writeln('          ',priorityvectors.riskvector[i]:8);
end;
writeln;
writeln(b,'          Consistency Ratio: ',CRvector[4]:6);
writeln;
write('          Based on the criteria of risk, your ');
writeln('comparisons were:');
WriteOptionComparisons(judgements.riskchoices,numoptions);
write('          To continue, press RETURN: ');
readln(hold);
GoTo 1;

```

```

[*****]
5:ClrScr;
  writeln(b,'          ATTRIBUTE - COST');
  writeln(b,'          ',conceptname);
  writeln(b,'    Option          Priority');
  writeln(b,'    -----          -----');
  writeln;
  for i := 1 to numoptions do
    begin
      write('          ',i,') ',options[i]:20);
      writeln('          ',priorityvectors.costvector[i]:8);
    end;
  writeln;
  writeln(b,'    Consistency Ratio: ',CRvector[5]:6);
  writeln;
  write('          Based on the criteria of cost, your ');
  writeln('comparisons were:');
  WriteOptionComparisons(judgements.costchoices,numoptions);
  write('          To continue, press RETURN: ');
  readln(hold);
  GoTo 1;
[*****]
6:ClrScr;
  writeln(b,'          FINAL PRIORITIES');
  writeln(b,'          -----');
  writeln;
  writeln(b,'          ',conceptname);
  writeln(b,'    Option          Priority');
  writeln(b,'    -----          -----');
  for i := 1 to numoptions do
    begin
      write('          ',i,') ',options[i]:20,' ');
      writeln(priorityvectors.finalpriorities[i]:8);
    end;
  writeln;
  write('          This is the final priority of the options ');
  writeln('for this concept,');
  write('          based upon your judgements of the importan');
  writeln('ce of the criteria');
  write('          and the ',numoptions,' options compared ag');
  writeln('ainst each other, with respect');
  writeln('          to the criteria. ');
  writeln;
  write(b,'To continue, press RETURN: ');
  readln(hold)
[*****]
  end; (with userdata do)
1:writeln
  end; (ViewPreviousRecord)

```



```

(*****
*****          INCLUDE FILE EXTRA3.DSS          *****
*****

```

```

overlay procedure CriteriaComparisons(Var comparisons: matrix;
                                     concept: Char20; Var selections: outputdatafile);

```

```

(CriteriaComparisons - procedure that actually compares the
criteria: performance, cost, schedule and risk.  Receives the
comparison matrix, the concept name and selections, a character
array to store the users choices.

```

The object code for this procedure is stored in dss.001.

Calls GetInfo, Compare

```

Variables - i,j: counting integers.
           Q1,Q2,A1,A2,hold: characters that receive data
           input.)

```

```

label 2;

```

```

Var i,j: integer;
    Q1,Q2,P,S,R,C,hold: char;

```

```

begin

```

```

  ClrScr;
  P := 'P';
  S := 'S';
  R := 'R';
  C := 'C';

```

```

2:for i := 1 to 4 do

```

```

  begin
    for j:= 1 to 4 do comparisons[i,j] := 1
    end;

```

```

  GetInfo(53,64); (comparison procedure explanation block 2)
  readln(hold);

```

```

  ClrScr;

```

```

  writeln('          Comparison of Criteria');

```

```

  writeln;

```

```

  writeln('          ',concept);

```

```

  writeln;

```

```

  writeln('          SCALE');

```

```

  writeln('          1 2 3 4 5 6 7 8 9');

```

```

  writeln('          |<----->| ');

```

```

  writeln('          equal      complete dominance');

```

```

  writeln;

```

```

  write(b,' Performance vs Schedule: ');

```

```

  readln(Q1,Q2);

```

```

  i := 1; j := 2;

```

```

  Compare(comparisons,P,S,Q1,Q2,i,j);

```

```

  selections[1] := Q1;

```

```

  selections[2] := Q2;

```

```

write(b,' Performance vs Risk: ');
readln(Q1,Q2);
j := 3;
Compare(comparisons,P,R,Q1,Q2,i,j);
selections[3] := Q1;
selections[4] := Q2;
write(b,' Performance vs Cost: ');
readln(Q1,Q2);
j := 4;
Compare(comparisons,P,C,Q1,Q2,i,j);
selections[5] := Q1;
selections[6] := Q2;
write(b,' Schedule vs Risk: ');
readln(Q1,Q2);
i := 2; j := 3;
Compare(comparisons,S,R,Q1,Q2,i,j);
selections[7] := Q1;
selections[8] := Q2;
write(b,' Schedule vs Cost: ');
readln(Q1,Q2);
j := 4;
Compare(comparisons,S,C,Q1,Q2,i,j);
selections[9] := Q1;
selections[10] := Q2;
write(b,' Risk vs Cost: ');
readln(Q1,Q2);
i := 3; j := 4;
Compare(comparisons,R,C,Q1,Q2,i,j);
selections[11] := Q1;
selections[12] := Q2;
end; (CriteriaComparisons)

```

```

overlay procedure GetPerformanceVector(Var inputfilerecord:
                                     userfile; Var quitindicator: integer);

```

(GetPerformanceVector - procedure that finds priorities of concept options given the criteria is performance.
Object code is stored in dss.001.

procedure Calls: GetInfo,AHP,CheckYorN, OptionComparisons)

label 2;

```

Var optionfile: file of conceptoption;
    optionrec: conceptoption;
    performancematrix: matrix;
    numoptions, i, sum: integer;
    answer: char;
    mean: real;

```

begin

```

Assign(optionfile,inputfilerecord.optiondata);
Reset(optionfile);
numoptions := FileSize(optionfile);

```

```

2:GetInfo(74,81);    (Performance Explanation)
write(b,b);
readln(answer);
CheckYorN(answer);
case answer of
  'Y','y':begin
    GetInfo(30,53); (AHP comparison scale)
    write(b,'Press RETURN to continue: ');
    readln(answer)
  end;
  'N','n': ;
end; (case of answer)
GetInfo(81,102);    (Performance Scale)
writeln;
write(b,'      To continue, press RETURN: ');
readln(answer);
ClrScr;
writeln;
writeln(b,'          ATTRIBUTE - PERFORMANCE');
writeln(b,'          ',inputfilerecord.conceptname);
writeln(b,'          Performance Scale');
writeln(b,'          1      2      3      4      5');
writeln(b,'          \Deficient    \Meets        \Exceeds');
writeln(b,'          Performance Requirements');
sum := 0;
write('      ');
for i := 1 to numoptions do
begin
  sum := sum + 1;
  Read(optionfile,optionrec);
  inputfilerecord.options[i] := optionrec.optionname;
  write('      ',i,' ') ,optionrec.optionname);
  if sum = 3 then
  begin
    writeln;
    write(b)
  end
end;
writeln;
Seek(optionfile,0);
writeln;
write(b);
for i := 1 to numoptions do write(' OPTION ',i,' ');
writeln;
write(' Coverage      ');
for i := 1 to numoptions do
begin
  Read(optionfile,optionrec);
  write('      ',optionrec.optionperformance.coverage);
  write('      ')
end;
writeln;
Seek(optionfile,0);
write(' Capacity      ');

```

```

for i := 1 to numoptions do
begin
  Read(optionfile,optionrec);
  write('      ',optionrec.optionperformance.capacity);
  write('      ')
end;
writeln;
Seek(optionfile,0);
write('    Quality      ');
for i := 1 to numoptions do
begin
  Read(optionfile,optionrec);
  write('      ',optionrec.optionperformance.quality);
  write('      ')
end;
writeln;
Seek(optionfile,0);
write('    Timeliness  ');
for i := 1 to numoptions do
begin
  Read(optionfile,optionrec);
  write('      ',optionrec.optionperformance.timeliness);
  write('      ')
end;
writeln;
Seek(optionfile,0);
write('    Availability ');
for i := 1 to numoptions do
begin
  Read(optionfile,optionrec);
  write('      ',optionrec.optionperformance.availability);
  write('      ')
end;
writeln;
Seek(optionfile,0);
write('    Survivability');
for i := 1 to numoptions do
begin
  Read(optionfile,optionrec);
  write('      ',optionrec.optionperformance.survivability);
  write('      ')
end;
writeln;
Seek(optionfile,0);
writeln;
with optionrec do
begin
  with optionperformance do
begin
    write('          MEAN      ');
    for i := 1 to numoptions do
begin
      sum := 0;
      mean := 0;

```

```

        Read(optionfile,optionrec);
        sum := coverage + capacity + quality + timeliness;
        sum := sum + availability + survivability;
        mean := sum/6;
        write(mean:6,' ');
    end;    ( for i )
    writeln
end        ( with optionperformance)
end;      ( with optionrec)
writeln;
with inputfilerecord do
begin
    OptionComparisons(performancematrix,numoptions,
                        judgements.performancechoices);
    AHP(performancematrix,priorityvectors.performancevector,
        numoptions,CRvector[2]);

    ClrScr;
    Seek(optionfile,0);
    writeln;
    writeln(b,'      ATTRIBUTE - PERFORMANCE');
    writeln(b,'      ',conceptname);
    writeln(b,'Option Name      Priority');
    writeln(b,'-----      -----');
    for i := 1 to numoptions do
    begin
        Read(optionfile,optionrec);
        write(' ');
        write(i,' ') ,optionrec.optionname:20,' ');
        writeln(priorityvectors.performancevector[i]:8)
    end;
    writeln;
    writeln(b,'      Consistency Ratio: ',CRvector[2]:6)
end;    (with inputfilerecord do)
writeln;
write('      This is the prioritized list of the ');
writeln('options based on');
write('      the criteria of performance.  If the ');
writeln('consistency ratio is');
write('      greater than 1.0E-01, a high degree of ');
writeln('inconsistency is');
write('      indicated in your comparisons.  Is it ');
write('acceptable (Y or N)? ');
readln(answer);
CheckYorN(answer);
if answer in ['N','n'] then
begin
    writeln;
    write(b,'Okay - we will do the comparisons over ');
    writeln('again. ');
    Delay(1500);
    Seek(optionfile,0);
    GoTo 2
end;
writeln;

```

```

write('      You may quit at this time and your inputs ');
writeln('will be saved for use at a ');
write('      later time. Q or q and RETURN to quit, or ');
write('RETURN only to continue: ');
readln(answer);
if (answer in ['Q','q']) then Quit(quitindicator);
Close(optionfile)
end; (GetPerformanceVector)

overlay procedure GetScheduleVector(Var inputfilerecord:
                                   userfile; quitindicator: integer);

( GetScheduleVector - finds priorities based on schedule)

label 2;

Var optionfile: file of conceptoption;
    optionrec: conceptoption;
    schedulematrix: matrix;
    i,j,numoptions,numti,sum: integer;
    answer: char;
    mean: real;

begin
    Assign(optionfile,inputfilerecord.optiondata);
    Reset(optionfile);
    numoptions := FileSize(optionfile);
    if numoptions > 5 then numoptions := 5;
    writeln;
2:GetInfo(102,112);
    writeln;
    write('      Do you wish to review the Comparison Scale ');
    write('(Y or N)? ');
    readln(answer);
    CheckYorN(answer);
    if (answer = 'Y') or (answer = 'y') then
    begin
        GetInfo(30,53);
        write(b,'To continue, press RETURN: ');
        readln(answer)
    end;
    ClrScr;
    writeln(b,' ATTRIBUTE - SCHEDULE');
    writeln(b,' ',inputfilerecord.conceptname);
    writeln(b,b,'      # Years');
    writeln('      Concept Option          SUM          MEAN');
    writeln('      -----          ---          ----');
    writeln;
    for i := 1 to numoptions do
    begin
        sum := 0;
        mean := 0;
        numti := 0;
        Read(optionfile,optionrec);

```

```

for j := 1 to 15 do
    if optionrec.optionti[j].tiname <> ''
        then numti := numti + 1;
for j := 1 to numti do sum :=
    sum + optionrec.optionti[j].tischedule;
mean := sum/numti;
write(' ',i,'') ',optionrec.optionname:20,' ');
writeln(sum:3,' ',mean:8)
end; (for j)
writeln;
write(' Would like to see a particular option ');
writeln('broken down');
write(' into its Technology Issues (Y or N)? ');
readln(answer);
CheckYorN(answer);
while (answer = 'Y') or (answer = 'y') do
begin
    sum := 0;
    mean := 0;
    writeln;
    write(' Please input the number of the option ');
    write('you wish to view: ');
    readln(answer);
    i := ord(answer) - 48;
    while not (i in [1..numoptions]) do
    begin
        write(' Number selected must be from 1 to ');
        writeln(numoptions,' as that is the number ');
        write(' of options for this concept. ');
        write('Please reselect: ');
        readln(answer);
        i := ord(answer) - 48
    end;
ClrScr;
Seek(optionfile,i-1);
Read(optionfile,optionrec);
with optionrec do
begin
    numti := 0;
    writeln;
    writeln(b,' ',optionname:20);
    writeln(b,b,b,' # Years to Solve');
    write(b,' Tech Issue NAME');
    writeln(b,'SCHEDULE');
    write(b,' -----');
    writeln(b,'-----');
    writeln;
    for j := 1 to 15 do
        if optionrec.optionti[j].tiname <> ''
            then numti := numti + 1;
    for j := 1 to numti do
    begin
        write(' ',j:2,'') ',optionti[j].tiname:45);
        writeln(' ',optionti[j].tischedule:2);

```

```

        sum := sum + optionti[j].tischedule
    end;
    mean := sum/numti;
    writeln;
    write(b,'SUM = ',sum:3,' ::: ');
    writeln('MEAN = ',mean:8)
end;
writeln;
write('      Do you wish to view another option ');
write('(Y or N)? ');
readln(answer);
CheckYorN(answer)
end;
ClrScr;
Seek(optionfile,0);
writeln(b,'      ATTRIBUTE - SCHEDULE');
writeln(b,'      ',inputfilerecord.conceptname);
writeln(b,b,'      # Years');
writeln('      Option Name          SUM          MEAN');
writeln('      -----          ---          ----');
writeln;
for i := 1 to numoptions do
begin
    sum := 0;
    mean := 0;
    numti := 0;
    Read(optionfile,optionrec);
    for j := 1 to 15 do
        if optionrec.optionti[j].tiname <> ''
            then numti := numti + 1;
    for j := 1 to numti do sum :=
        sum + optionrec.optionti[j].tischedule;
    mean := sum/numti;
    write(' ',i,' ') ,optionrec.optionname:20,' ');
    writeln(sum:3,' ',mean:8)
end;
writeln;
with inputfilerecord do
begin
    OptionComparisons(schedulmatrix,numoptions,
        judgements.schedulechoices);
    AHP(schedulmatrix,priorityvectors.schedulevector,
        numoptions,CRvector[3]);

    ClrScr;
    writeln(b,'      ATTRIBUTE - SCHEDULE');
    writeln(b,'      ',concept);
    writeln;
    writeln('      Option Name          Priority');
    writeln('      -----          -----');
    writeln;
    Seek(optionfile,0);
    for i := 1 to numoptions do
        begin
            Read(optionfile,optionrec);

```



```

        write(' ',i,'')',optionrec.optionname:20,' ');
        writeln(priorityvectors.schedulevector[i]:8)
    end;
    writeln;
    writeln('          Consistency Ratio: ',CRvector[3]:6)
end; (with inputfilerecord do)
writeln;
write('          This is the priority vector of the ');
writeln('options based upon');
write('          the criteria of schedule and your ');
writeln('compairisons. The Consistency');
write('          ratio should be less than 1.0E-01 to ');
writeln('indicate consistent comparisons. ');
write('          Is it acceptable (Y or N)? ');
readln(answer);
CheckYorN(answer);
if (answer in ['N','n']) then
begin
    writeln;
    write('          Okay - we will do the comparisons over ');
    writeln('again. ');
    Delay(1500);
    Seek(optionfile,0);
    GoTo 2
end;
Close(optionfile);
writeln;
write('          You may quit at this time and your inputs ');
writeln('will be saved for use at a ');
write('          later date. Q or q and RETURN to quit, or ');
write('RETURN only to continue: ');
readln(answer);
if (answer in ['Q','q']) then Quit(quitindicator)
end; (GetScheduleVector)

overlay procedure GetRiskVector(Var inputfilerecord: userfile;
                                Var quitindicator: integer);

(GetRiskVector - finds the priority vector for given concept
options based upon the risk criteria.)

label 2;

Var optionfile: file of conceptoption;
    optionrec: conceptoption;
    riskmatrix: matrix;
    numoptions, numti, i, j, sum: integer;
    mean, sigma: real;
    answer: char;

begin
    Assign(optionfile,inputfilerecord.optiondata);
    Reset(optionfile);
    numoptions := FileSize(optionfile);

```

```

GetInfo(112,121);      (Risk explanation)
writeln;
write('      Do you wish to review the Comparison Scale ');
write(' (Y or N)? ');
readln(answer);
CheckYorN(answer);
if (answer = 'Y') or (answer = 'y') then
begin
  GetInfo(30,53);
  write('      To continue, press RETURN: ');
  readln(answer)
end;
GetInfo(121,139);      (Risk Scale)
writeln;
write(b,'To continue, press RETURN: ');
readln(answer);
2:ClrScr;
writeln(b,'      ATTRIBUTE - RISK ');
write(b,'      ');
writeln(inputfilererecord.conceptname);
write('      Concept Option          Mean      Std');
writeln(' Deviation');
write('      -----          ----      ---');
writeln('-----');
writeln;
for i := 1 to numoptions do
begin
  sum := 0;
  mean := 0;
  sigma := 0;
  numti := 0;
  Read(optionfile,optionrec);
  for j := 1 to 15 do
    if optionrec.optionti[j].tiname <> ''
      then numti := numti + 1;
  for j := 1 to numti do sum := sum +
    optionrec.optionti[j].tirisk;
  mean := sum/numti;
  for j := 1 to numti do sigma :=
    sigma + sqr((optionrec.optionti[j].tirisk - mean));
  sigma := sqrt(sigma);
  write(' ',i,' ',optionrec.optionname:20,' ');
  writeln(mean:8,' ',sigma:10)
end;
writeln;
writeln('      Would like to see a particular option broken');
write('      down into its Tech Issues (Y or N)? ');
readln(answer);
CheckYorN(answer);
while (answer in ['Y','y']) do
begin
  writeln;
  write('      Please input the number of the option you ');
  write('wish to view: ');

```

```

readln(answer);
i := ord(answer) - 48;
while not (i in [1..numoptions]) do
begin
  write('      Number selected is not between 1 and ');
  writeln(numoptions,',');
  write('      the number of options associated with ');
  writeln('this concept. ');
  write('      Please reselect another entry: ');
  readln(answer);
  i := ord(answer) - 48
end;
ClrScr;
writeln(b,'      ATTRIBUTE - RISK');
Seek(optionfile,i-1);
Read(optionfile,optionrec);
sum := 0;
mean := 0;
sigma := 0;
numti := 0;
with optionrec do
begin
  writeln(b,'      OPTION - ',optionname);
  writeln;
  for j := 1 to 15 do
    if optionrec.optionti[j].tiname <> ''
      then numti := numti + 1;
  write(b,'      Technology Issue Name');
  writeln(b,'Risk');
  write(b,'      -----');
  writeln(b,'----');
  for j := 1 to numti do
  begin
    write(' ',j:2,' ') ,optionti[j].tiname:45,' ');
    writeln(optionti[j].tirisk);
    sum := sum + optionti[j].tirisk
  end;
  mean := sum/numti;
  for j := 1 to numti do sigma :=
    sigma + sqr((optionti[j].tirisk - mean));
  sigma := sqrt(sigma);
  writeln;
  write('      MEAN = ',mean:8,' ');
  writeln('STD DEVIATION = ',sigma:8)
end;
write('      Below is a breakout of the # of Tech Issues');
writeln(' in each risk catagory. ');
writeln;
for i := 1 to 5 do
begin
  sum := 0;
  for j := 1 to numti do
  begin
    if optionrec.optionti[j].tirisk = 1

```

```

                                then sum := sum + 1
                                end;
                                write('          ',sum)
                                end;
                                writeln;
                                write('          Very Low          Low          Medium          High');
                                writeln('          Very High');
                                writeln;
                                write('          Do you wish to view another option ');
                                write('(Y or N)? ');
                                readln(answer);
                                CheckYorN(answer)
                                end;
                                Seek(optionfile,0);
                                ClrScr;
                                writeln(b,'          ATTRIBUTE - RISK ');
                                write(b,'          ');
                                writeln(inputfilerecord.conceptname);
                                write('          Concept Option          Mean          Std');
                                writeln(' Deviation');
                                write('          -----          ----          ---');
                                writeln('-----');
                                writeln;
                                for i := 1 to numoptions do
                                begin
                                    sum := 0;
                                    mean := 0;
                                    sigma := 0;
                                    numti := 0;
                                    Read(optionfile,optionrec);
                                    for j := 1 to 15 do
                                        if optionrec.optionti[j].tiname <> ''
                                            then numti := numti + 1;

                                    for j := 1 to numti
                                        do sum := sum + optionrec.optionti[j].tirisk;
                                    mean := sum/numti;
                                    for j := 1 to numti do sigma :=
                                        sigma + sqrt((optionrec.optionti[j].tirisk - mean));
                                    sigma := sqrt(sigma);
                                    write(' ',i:2,' ') ,optionrec.optionname:20,' ');
                                    writeln(mean:8,' ',sigma:10)
                                end;
                                writeln;
                                with inputfilerecord do
                                begin
                                    OptionComparisons(riskmatrix,numoptions,
                                                            judgements.riskchoices);
                                    AHP(riskmatrix,priorityvectors.riskvector,numoptions,
                                                            CRvector[4]);

                                    Seek(optionfile,0);
                                    ClrScr;
                                    writeln(b,'          ATTRIBUTE - RISK');
                                    writeln(b,'          ',conceptname);
                                    writeln(b,'          Option          Priority');
                                    writeln(b,'          -----          -----');

```

```

writeln;
for i := 1 to numoptions do
begin
  Read(optionfile,optionrec);
  write('      ',i,' ') ,optionrec.optionname:20);
  writeln('      ',priorityvectors.riskvector[i]:8);
end;
writeln;
writeln('      Consistency Ratio: ',CRvector[4]:6)
end; (with inputfilerecord)
writeln;
write('      Consistency Ratio should be below 1.0E-01, ');
writeln('otherwise some comparisons ');
write('      are inconsistent. Do you agree with this ');
write('priority listing (Y or N)? ');
readln(answer);
CheckYorN(answer);
if answer in ['N','n'] then
begin
  writeln(b,'Okay, we will do the comparisons again. ');
  Seek(optionfile,0);
  Delay(1500);
  GoTo 2
end;
Close(optionfile);
writeln;
write('      You may quit at this time and your inputs ');
writeln('will be saved for use at a ');
write('      later date. Q or q and RETURN to quit, or ');
write('RETURN only to continue: ');
readln(answer);
if answer in ['Q','q'] then Quit(quitindicator)
end; (GetRiskVector)

overlay procedure GetCostVector(Var inputfilerecord: userfile;
                               Var quitindicator: integer);

(GetCostVector - finds the option priority vector based on the
cost criteria.)

label 2;

Var optionfile: file of conceptoption;
    optionrec: conceptoption;
    costmatrix: matrix;
    i, j, numoptions, numti: integer;
    answer: char;
    sum, mean: real;

begin
  Assign(optionfile,inputfilerecord.optiondata);
  Reset(optionfile);
  numoptions := FileSize(optionfile);
  writeln;
  GetInfo(139,149);

```

```

writeln;
write('      Do you wish to review the Comparison Scale ');
write('(Y or N)? ');
readln(answer);
CheckYorN(answer);
if (answer = 'Y') or (answer = 'y') then
begin
  GetInfo(30,53);
  write('      To continue, press RETURN: ');
  readln(answer)
end;
2:ClrScr;
writeln(b,'      ATTRIBUTE - COST ');
write(b,'      ');
writeln(inputfilerrecord.conceptname);
write('      Concept Option      TOTAL COST - $');
writeln('      # Tech Issues');
write('      -----');
writeln('      -----');
writeln;
for i := 1 to numoptions do
begin
  sum := 0;
  numti := 0;
  Read(optionfile,optionrec);
  for j := 1 to 15 do
    if optionrec.optionti[j].tiname <> ''
      then numti := numti + 1;
  for j := 1 to numti do sum :=
    sum + optionrec.optionti[j].ticost;
  write('      ',i,',') ',optionrec.optionname:20,' ');
  writeln(sum:8,'      ',numti:2)
end;
writeln;
Seek(optionfile,0);
write('      Would like to see a particular option ');
writeln('broken down');
write('      into its Tech Issues (Y or N)? ');
readln(answer);
CheckYorN(answer);
while (answer in ['Y','y']) do
begin
  writeln;
  write('      Please input the number of the option you ');
  write('wish to view: ');
  readln(answer);
  i := ord(answer) - 48;
  while not (i in [1..numoptions]) do
begin
  write('      Number selected is not between 1 and ');
  writeln(numoptions,',');
  write('      the number of options associated with');
  writeln(' this concept. ');
  write('      Please reselect: ');

```

```

        readln(answer);
        i := ord(answer) - 48
    end;    (while i not in 1 to numoptions)
ClrScr;
Seek(optionfile,i-1);
Read(optionfile,optionrec);
writeln(b,'      ATTRIBUTE - COST');
with optionrec do
begin
    writeln(b,'      OPTION - ',optionname);
    writeln;
    sum := 0;
    mean := 0;
    numti := 0;
    for j := 1 to 15 do
        if optionrec.optionti[j].tiname <> ''
            then numti := numti + 1;
        writeln(b,' Technology Issue Name',b,'      $ - Cost');
        writeln(b,' -----',b,' -----');
        writeln;
        for j := 1 to numti do
            begin
                write(' ',j:2,' ') ,optionti[j].tiname:45,'      ');
                writeln(optionti[j].ticoast:8);
                sum := sum + optionti[j].ticoast
            end;
        mean := sum/numti;
        writeln;
        write('      :::      SUM = ',sum:8,'      ::: ');
        writeln('MEAN = ',mean:8,'      :::');
    end;
    writeln;
    write('      Do you wish to view another option ');
    write('(Y or N)? ');
    readln(answer);
    CheckYorN(answer)
end;
Seek(optionfile,0);
ClrScr;
writeln(b,b,'ATTRIBUTE - COST');
writeln(b,b,inputfilerecord.conceptname);
write('      Concept Option      TOTAL COST - $');
writeln('      # Tech Issues');
write('      -----');
writeln('      -----');
writeln;
for i := 1 to numoptions do
begin
    sum := 0;
    numti := 0;
    Read(optionfile,optionrec);
    for j := 1 to 15 do
        if optionrec.optionti[j].tiname <> ''
            then numti := numti + 1;

```

```

    for j := 1 to numti do sum :=
        sum + optionrec.optionti[j].ticost;
    write(' ',i,' ') ,optionrec.optionname:20,' ');
    writeln(sum:8,' ',numti:2)
end;
writeln;
with inputfilerecord do
begin
    OptionComparisons(costmatrix,numoptions,
        judgements.costchoices);
    AHP(costmatrix,priorityvectors.costvector,
        numoptions,CRvector[5]);

    Seek(optionfile,0);
    ClrScr;
    writeln(b,'    ATTRIBUTE - COST');
    writeln('          Option          Priority');
    writeln('          -----          -----');
    writeln;
    for i := 1 to numoptions do
    begin
        Read(optionfile,optionrec);
        write(' ',i,' ') ,optionrec.optionname:20;
        writeln(' ',priorityvectors.costvector[i]:8);
    end;
    writeln;
    writeln('    Consistency Ratio: ',CRvector[5]:6)
end; (with inputfilerecord)
writeln;
writeln('    If Consistency Ratio is above 1.0E-01, some');
write('    inconsistency in the pairwise judgements is');
writeln(' indicated');
write('    Do you agree with this priority listing ');
write('(Y or N)? ');
readln(answer);
CheckYorN(answer);
if answer in ['N','n'] then
begin
    writeln(b,'Okay, we will do the comparisons again. ');
    Delay(1500);
    Seek(optionfile,0);
    GoTo 2
end;
Close(optionfile)
end; (GetCostVector)

```



```

(*****
*****      EXTRA4.DSS INCLUDE FILE      *****
*****)

```

```

procedure PrintWholeRecord(Var userdata: userfile);

```

```

(PrintWholeRecord - prints an entire record of type userfile,
the user database main component. Accomplishes this
by calling PrintPreviousRun a number of times.)

```

```

Var i: integer;

```

```

begin

```

```

    writeln(Lst,' ');
    writeln(Lst,b,b,'AFSTC DECISION SUPPORT SYSTEM');
    writeln(Lst,b,b,'      REPORT OF');
    writeln(Lst,b,b,'    PRIORITIZATION PROCESS');
    for i := 1 to 3 do writeln(Lst,b,' ');
    PrintPreviousRun(userdata,1);
    for i := 1 to 4 do writeln(Lst,b,' ');
    PrintPreviousRun(userdata,2);
    for i := 1 to 5 do writeln(Lst,b,' ');
    writeln(Lst,b,b,'      1');
    for i := 1 to 8 do writeln(Lst,b,' ');
    PrintPreviousRun(userdata,3);
    for i := 1 to 6 do writeln(Lst,b,' ');
    PrintPreviousRun(userdata,4);
    for i := 1 to 7 do writeln(Lst,b,' ');
    writeln(Lst,b,b,'      2');
    for i := 1 to 8 do writeln(Lst,b,' ');
    PrintPreviousRun(userdata,5);
    for i := 1 to 5 do writeln(Lst,b,' ');
    PrintPreviousRun(userdata,6);
    for i := 1 to 3 do writeln(Lst,b,' ');
    write(Lst,b,'This completes this listing from the ');
    writeln(Lst,'AFSTC DSS');
    writeln(Lst,' ');
    writeln(Lst,' ');
    writeln(Lst,b,b,'      3')
end; (PrintWholeRecord)

```

```

procedure Reports(personalfile: Char12; status,numrecords:
integer);

```

```

( Reports - Governs the user database hardcopy reports section)

```

```

label 2;

```

```

Var userdata: file of userfile;
    userrecord: userfile;
    answer: char;
    integeranswer,i: integer;

```

```

begin
  Assign(userdata,personalfile);
  Reset(userdata);
2:ClrScr;
  writeln;

  writeln(b,'          PRINT OPTIONS');
  writeln(b,'          -----');
  writeln(b,'          1) PRINT entire record');
  writeln(b,'              [1 to ',numrecords,']');
  writeln(b,'          2) Return to previous menu');
  writeln;
  write(b,'          SELECTION: ');
  readln(answer);
  while not (answer in ['1','2']) do
    begin
      write('          ',answer,' not an option, please ');
      write('reselect: ');
      readln(answer)
    end;
  case answer of
    '1': begin
      writeln;
      write('          Please select the program run you wish');
      writeln(' to PRINT by its number. ');
      write('          For example, 1 for the first program ');
      write('run: ');
      readln(answer);
      integeranswer := ord(answer) - 48;
      while not (integeranswer in [1..numrecords]) do
        begin
          write('          Selection not in file, please ');
          write('reselect number: ');
          readln(answer);
          integeranswer := ord(answer) - 48
        end;
      Seek(userdata,integeranswer-1);
      Read(userdata,userrecord);
      PrintWholeRecord(userrecord);
      (will use two existing files)

      GoTo 2
    end;
    '2': ;
  end; (case answer of 1 or 2)
  Close(userdata)
end; (View)

procedure View (personalfile: Char12; status,numrecords:
                                     integer);

( View - procedure to allow the user to view a record in the
  users personal file.)

label 1,2,3;

```

```

Var userdata: file of userfile;
    userrecord: userfile;
    answer: char;
    integeranswer,i: integer;

begin
    Assign(userdata,personalfile);
    Reset(userdata);
2:ClrScr;
    writeln;
    writeln(b,'          VIEW OPTIONS');
    writeln(b,'          -----');
    writeln;
    writeln(b,'      1) Select a record to VIEW');
    writeln(b,'          [1 to ',numrecords,']');
    writeln(b,'      2) Return to previous menu');
    writeln;
    write(b,'          SELECTION: ');
    readln(answer);
    while not (answer in ['1','2']) do
        begin
            write('          ',answer,' not an option, please ');
            write('reselect: ');
            readln(answer)
        end;
    case answer of
        '1': begin
            writeln;
            write('      Please select the program run you wish');
            writeln(' to view by its number. ');
            write('      For example, 1 for the first program ');
            write('run: ');
            readln(answer);
            integeranswer := ord(answer) - 48;
            while not (integeranswer in [1..numrecords]) do
                begin
                    write('      Selection not in file, please ');
                    write('reselect number: ');
                    readln(answer);
                    integeranswer := ord(answer) - 48
                end
            end;
        '2': GoTo 1
    end; {case answer of 1 or 2}
    Seek(userdata,integeranswer-1);
    Read(userdata,userrecord);
3:ClrScr;
    writeln(b,'      ',userrecord.conceptname);
    writeln;
    writeln(b,'      SECTIONS TO VIEW');
    writeln(b,'      -----');
    writeln;
    writeln(b,'      1) Criteria');
    writeln(b,'      2) Performance');

```

```

writeln(b,' 3) Schedule');
writeln(b,' 4) Risk');
writeln(b,' 5) Cost');
writeln(b,' 6) Final priorities');
writeln(b,' 7) Return to previous menu');

writeln;
write('      The catagories above refer to the sections ');
writeln('of the program');
write('      in which prioritization was performed. ');
writeln('Select the number');
write('      of the section you wish to VIEW: ');
readln(answer);
if answer = '7' then GoTo 2;
integeranswer := ord(answer) - 48;
while not (answer in ['1'..'6']) or
          not (integeranswer in [1..status]) do
begin
  if not (answer in ['1'..'6']) then
    write('      Please reselect: ')
  else
    begin
      write('      That section of this program run ');
      write('is not completed. Please reselect: ')
    end; (else)
  readln(answer);
  if answer = '7' then GoTo 2;
  integeranswer := ord(answer) - 48
end; (while answer not in 1 to 7)
writeln;
if integeranswer in [1..6] then
begin
  ViewPreviousRun(userrecord,integeranswer);
  GoTo 3
end; (if then clause)
1:Close(userdata)
end; (View)

procedure DatabaseManagement(Var quitindicator: integer);

(DatabaseManagement - main database management procedure.
this procedure comprises the heart of the database management
system for the pilot DSS.)

label 1,2,3;

Var  allconcepts: file of datafile;
     optionfile: file of conceptoption;
     temprecord, newconceptrecord: datafile;
     tempfilename, optionfilename: Char20;
     i,j: integer;
     answer, tempchar: char;

begin
  Assign(allconcepts,'concepts.dss');

```

```

Reset(allconcepts);
2:ClrScr;
writeln;
writeln;
writeln(b,'    CONCEPT DATABASE MANAGEMENT SYSTEM');

writeln;
writeln(b,'          DATABASE OPTIONS');
writeln(b,'          -----');
writeln(b,'          1) ENTER a new concept');
writeln(b,'          2) CHANGE or ADD data to');
writeln(b,'              a concept');
writeln(b,'          3) ERASE a current concept');
writeln(b,'          4) CONTINUE program');
writeln(b,'          5) QUIT program ');
writeln;
write(b,'          SELECTION: ');
readln(answer);
if not (answer in ['1','2','3','4','5']) then
begin
    writeln;
    write('          ',answer,' is an incorrect selection. ');
    write('Please reselect: ');
    readln(answer)
end;
case answer of
    '1': begin
        tempchar := answer;
        if FileSize(allconcepts) = maxnumconcepts then
            begin
                ClrScr;
                writeln;
                write('          The maximum number of concepts ');
                writeln('allowed in the database is 10. ');
                write('          A concept currently in the data');
                writeln('base will have to be erased before');
                writeln('          new data can be added. ');
                GoTo 2
            end;
        (if more than 10 concepts)
        for i := 1 to 5 do write('          WARNING');
        writeln;
        write('          If you do not have all of the data ');
        writeln('to input at least two full concept ');
        write('          options into the database, it is best');
        writeln(' to wait until you do. Otherwise, ');
        write('          the program may crash with incomplete');
        writeln(' data files during execution. Do ');
        write('          you wish to continue at this time (Y ');
        write('or N)? ');
        readln(answer);
        if answer in ['N','n'] then GoTo 2;
        InputNewConcept(newconceptrecord);
        Seek(allconcepts,FileSize(allconcepts));
        Write(allconcepts,newconceptrecord);
        case FileSize(allconcepts) of

```

```

1: optionfilename := 'concepta.dss';
2: optionfilename := 'conceptb.dss';
3: optionfilename := 'conceptc.dss';
4: optionfilename := 'conceptd.dss';
5: optionfilename := 'concepte.dss';

6: optionfilename := 'conceptf.dss';
7: optionfilename := 'conceptg.dss';
8: optionfilename := 'concepth.dss';
9: optionfilename := 'concepti.dss';
10: optionfilename := 'conceptj.dss'
end; (case to decide filename)
InputConceptOptions(optionfilename,tempchar);
GoTo 2
end; (case answer of enter new concept)
'2': begin
    tempchar := answer;
    ClrScr;
    write('          Select letter of concept ');
    writeln('to add or change data. ');
    writeln;
    write(b,'          Concept');
    write(b,'          -----');
    Seek(allconcepts,0);
    for i := 1 to FileSize(allconcepts) do
    begin
        Read(allconcepts,newconceptrecord);
        write(b,'          ',chr(i+64),' ');
        writeln(newconceptrecord.conceptname)
    end;
    writeln;
    write(b,'          SELECTION: ');
    readln(answer);
    if answer in ['a'..'j'] then
        answer := chr(ord(answer)-32);
    while not
    (answer in ['A'..chr(FileSize(allconcepts)+64)]) do
    begin
        write(b,'Incorrect answer, Please Reselect: ');
        readln(answer)
    end;
    case answer of
        'A': optionfilename := 'concepta.dss';
        'B': optionfilename := 'conceptb.dss';
        'C': optionfilename := 'conceptc.dss';
        'D': optionfilename := 'conceptd.dss';
        'E': optionfilename := 'concepte.dss';
        'F': optionfilename := 'conceptf.dss';
        'G': optionfilename := 'conceptg.dss';
        'H': optionfilename := 'concepth.dss';
        'I': optionfilename := 'concepti.dss';
        'J': optionfilename := 'conceptj.dss'
    end; (case of answer)
    InputConceptOptions(optionfilename,tempchar);
    GoTo 2
end;

```

```

end; (case of add or change data to existing file)
'3': begin
  ClrScr;
  write('          Select letter of concept ');
  writeln('to ERASE. ');
  writeln;
  writeln(b, '          Concept ');
  writeln(b, '          ----- ');
  Seek(allconcepts, 0);
  for i := 1 to FileSize(allconcepts) do
  begin
    Read(allconcepts, newconceptrecord);
    write(b, '          ', chr(i+64), ' ');
    writeln(newconceptrecord.conceptname)
  end;
  writeln(b, '          ', chr(i+65), ' None ');
  writeln;
  write(b, '          SELECTION: ');
  readln(answer);
  if answer in ['a'..'j'] then
    answer := chr(ord(answer)-32);
  while not
    (answer in ['A'..chr(FileSize(allconcepts)+65)]) do
  begin
    write(b, 'Incorrect answer, Please Reselect: ');
    readln(answer);
    if answer in ['a'..'j'] then
      answer := chr(ord(answer)-32)
    end;
  end;
  tempchar := answer;
  writeln;
  Seek(allconcepts, ord(answer)-64);
  Read(allconcepts, newconceptrecord);
  write(b, 'Confirm ERASE of ');
  write(newconceptrecord.conceptname, ' (Y or N)? ');
  readln(answer);
  CheckYorN(answer);
  if answer in ['N', 'n'] then GoTo 2;
  answer := tempchar;
  case answer of
    'A': optionfilename := 'concepta.dss';
    'B': optionfilename := 'conceptb.dss';
    'C': optionfilename := 'conceptc.dss';
    'D': optionfilename := 'conceptd.dss';
    'E': optionfilename := 'concepte.dss';
    'F': optionfilename := 'conceptf.dss';
    'G': optionfilename := 'conceptg.dss';
    'H': optionfilename := 'concepth.dss';
    'I': optionfilename := 'concepti.dss';
    'J': optionfilename := 'conceptj.dss';
    'K': GoTo 2
  end; (case of answer)
  for i := (ord(answer)-64) to
    (FileSize(allconcepts) - 1) do
  begin

```

```

        Seek(allconcepts,i+1);
        Read(allconcepts,newconceptrecord);
        Seek(allconcepts,i);
        Write(allconcepts)
    end;

    Seek(allconcepts,FileSize(allconcepts)-1);
    newconceptrecord.conceptname := '';
    newconceptrecord.conceptorigin := '';
    Write(allconcepts,newconceptrecord)
end; (case of erase a concept - choice 3)
'4': ;
'5': begin
    Quit(quitindicator);
    if quitindicator <> 1 then GoTo 2
    end
end; (case answer of)
1:Close(allconcepts)
end; (DatabaseManagement)

```



```

(*****
*****      INCLUDE FILE EXTRAS.DSS      *****
*****

```

```

procedure GetUserFileName(Var personalfile: Char12;
                          Var status, quitindicator: integer);

```

```

(GetUserFileName - Gets the name of the user for access into the
user database (stores the name in personalfile, a 12
character string.) This procedure also queries for
access to the main concept database.)

```

```

label 2;

```

```

Var answer: char;
    i: integer;
    outfilename,temp: outputfilename;

```

```

begin

```

```

2:answer := 'N';

```

```

while answer in ['N','n'] do

```

```

begin

```

```

    personalfile := ' ';

```

```

    ClrScr;

```

```

    for i := 1 to 8 do outfilename[i] := ' ';

```

```

    outfilename[9] := '.';

```

```

    outfilename[10] := 'd';

```

```

    outfilename[11] := 's';

```

```

    outfilename[12] := 's';

```

```

    for i := 1 to 5 do writeln;

```

```

    write('      Enter first initial: ');

```

```

    readln(outfilename[1]);

```

```

    write('      Enter middle initial (or second letter');

```

```

    write(' of first name): ');

```

```

    readln(outfilename[2]);

```

```

    write('      Enter first six (6) letters of last name: ');

```

```

    read(outfilename[3],outfilename[4],outfilename[5],
        outfilename[6],outfilename[7],outfilename[8]);

```

```

    writeln;

```

```

    write('      ',outfilename[1],' ',outfilename[2],' ');

```

```

    for i := 3 to 8 do write(outfilename[i]);

```

```

    write('      Is this correct (Y or N)? ');

```

```

    readln(answer);

```

```

    CheckYorN(answer)

```

```

end; (while answer is No)

```

```

temp := outfilename;

```

```

for i := 1 to 8 do

```

```

    while not (ord(outfilename[i]) in [65..90,97..122]) do
        outfilename[i] := 'z';

```

```

personalfile := outfilename;

```

```

writeln;

```

```

write('      Is this your first time using this program');

```

```

write(' (Y or N)? ');

```

```

readln(answer);

```

```

CheckYorN(answer);
if answer in ['Y','y'] then
begin
  if Exist(personalfile) = false then status := 0
  else
  begin
    write('      WARNING: A file exists under the name ');
    writeln('you have given. ');
    write('      It will be erased if you do not change ');
    writeln(' your answer. ');
    write('      Do you wish to Change (Y or N -- ');
    write('last answer [Y])? ');
    readln(answer);
    CheckYorN(answer);
    if answer in ['Y','y'] then status := 0
    else status := 10
  end (else Exist is true)
end (if answer is Yes)
else
begin
  if Exist(personalfile) = true then status := 10 else
  begin
    write('      There is no record of you using this ');
    writeln('program before under ');
    write(b,'the name you have given: ');
    write(temp[1], ' ', temp[2], ' ');
    for i := 3 to 8 do write(temp[i]);
    writeln('. ');
    write('      You may change the name you have given ');
    writeln('or you may start a new file. ');
    write('      Do you wish to input a new name ');
    write('(Y or N)? ');
    readln(answer);
    CheckYorN(answer);
    if answer in ['y','Y'] then GoTo 2 else status := 0
  end (if file does not exist)
end; (if answer is Yes)
write('      Do you want to enter the database and input ');
write('new data (Y or N)? ');
readln(answer);
CheckYorN(answer);
if answer in ['Y','y'] then
      DatabaseManagement(quitindicator)
end; (GetUserFileName)

overlay procedure Review(Var personalfile: Char12;
      Var status,quitindicator: integer);

(Review is the main user database file. It governs all of the
functions of viewing, printing, or deleting records from the
user database. Object code for this file goes to dss.002)

label 2;

```

```

Var userdata: file of userfile;
    userrecord: userfile;
    i,j,numrecords: integer;
    answer: char;

begin
2:Assign(userdata,personalfile);
  Reset(userdata);
  Seek(userdata,FileSize(userdata)-1);
  Read(userdata,userrecord);
  numrecords := FileSize(userdata);
  ClrScr;
  if status = 5 then
    begin
      writeln;
      write('          You have ',numrecords,' previously ');
      writeln('completed program runs. ');
      writeln
    end
  else
    begin
      writeln;
      write('          You have ',numrecords-1,' previously');
      writeln(' completed program runs, and');
      write('          1 incomplete run, # ',numrecords);
      writeln(', and you were working on ');
      writeln('          ',userrecord.conceptname,'. ')
    end;
  writeln;
  writeln(b,'          REVIEW OPTIONS');
  writeln(b,'          -----');
  writeln(b,'          1) View a previous run');
  writeln(b,'          2) Print out any run');
  writeln(b,'          3) Start new run');
  writeln(b,'          4) Continue unfinished run');
  writeln(b,'          5) Erase all previous runs');
  writeln(b,'          6) Quit DSS');
  writeln;
  write(b,'Please input the number of your choice: ');
  readln(answer);
  while not ord(answer) in [49..55] do
    begin
      write('          Must be a number between 1 and 6. ');
      write('Please reselect: ');
      readln(answer)
    end;
  while (answer = '4') and not (status in [1..4]) do
    begin
      write('          You do not have any unfinished runs. ');
      write('Please reselect: ');
      readln(answer)
    end;
  case answer of
    '1': begin
      Close(userdata);

```

```

        View(personalfile,status,numrecords);
        GoTo 2
    end;
'2': begin
    Close(userdata);
    Reports(personalfile,status,numrecords);
    GoTo 2
end;
'3': begin
    Close(userdata);
    if status = 5 then status := 6 else status := 7;
    InitializeUserFile(personalfile,status);
    status := 0
end;
'4': begin
    Close(userdata);
    case status of
        1: begin
            write(b,'You quit after choosing the ');
            writeln('criteria values. You will ');
            write(b,'start with compairing the ');
            writeln('options based upon performance.')
        end;
        2: begin
            write(b,'You quit after choosing the ');
            writeln('performance values. You will ');
            write(b,'start with compairing the ');
            writeln('options based upon schedule.')
        end;
        3: begin
            write(b,'You quit after choosing the ');
            writeln('schedule values. You will ');
            write(b,'start with compairing the ');
            writeln('options based upon risk.')
        end;
        4: begin
            write(b,'You quit after choosing the ');
            writeln('risk values. You will ');
            write(b,'start with compairing the ');
            writeln('options based upon cost.')
        end
    end; (case of status)
    write(b,'Press RETURN to continue: ');
    readln(answer)
end; (answer being 4 - continue unfinished run)
'5': begin
    writeln;
    write(b,'Confirm erase of all previous runs ');
    write('(Y or N)? ');
    readln(answer);
    CheckYorN(answer);
    if answer in ['Y','y'] then
        begin
            writeln;

```

```

        Seek(userdata,FileSize(userdata)-1);
        Read(userdata,userrecord);
        Close(userdata);
        Erase(userdata);
        status := 0;
        InitializeUserFile(personalfile,status);
        Reset(userdata);
        Write(userdata,userrecord);
        Close(userdata);
        status := 10;
        InitializeUserFile(personalfile,status);
        write('          All previous records erased ');
        writeln('except for the most recent run. ');
        Delay(1000);
        GoTo 2
      end (if answer is Yes)
    else GoTo 2
  end; (case 5)
'6': begin
        Close(userdata);
        Quit(quitindicator);
        if quitindicator = 0 then GoTo 2
      end (case 6)
  end (case of answer)
end; (Review)

```

```

overlay procedure ListConcepts(Var concept: Char20; Var
                               filename: Char12; Var quitindicator: integer);

```

(ListConcepts - Opens the file concepts.dss to obtain all of the concepts in the database. Displays them for selection by the user.)

```

label 2;

```

```

Var choicefile: file of datafile;
    conceptrec: datafile;
    numconcepts,level,filenbr,i: integer;
    choice: char;
    origin: Char20;

```

```

begin
  Assign(choicefile,'concepts.dss');
2:Reset(choicefile);
  filenbr := 0;
  numconcepts := FileSize(choicefile);
  ClrScr;
  writeln;
  writeln(b,'          CONCEPT          ORIGIN');
  writeln(b,'          -----          -----');
  writeln;
  for i := 1 to numconcepts do
    begin
      with conceptrec do

```

```

begin
  read(choicefile,conceptrec);
  write(' ');
  write(chr(i+64),') ',conceptname:20);
  writeln(' ',conceptorigin:20)
end (with conceptrec)
end; (for i)
writeln;
write(' Please choose the letter of the concept ');
writeln('you wish to work with. ');
writeln;
write(b,' ? will list available help: ');
readln(choice);
filenbr := ord(choice);
if (filenbr in [97..(numconcepts+96)]) then
  filenbr := filenbr - 32;
while not
  (filenbr in [63,65..(numconcepts+64)]) do
begin
  writeln;
  write(' You have not choosen a viable option. ');
  write('Please reselect: ');
  readln(choice);
  filenbr := ord(choice);
  if (filenbr in [97..(numconcepts+96)]) then
    filenbr := filenbr - 32;
  end; (while filenbr not an option)
case filenbr of
65..74: begin
  Seek(choicefile,filenbr-65);
  Read(choicefile,conceptrec);
  concept := conceptrec.conceptname;
  origin := conceptrec.conceptorigin;
  Close(choicefile);
  case filenbr of
    65:filename := 'concepta.dss';
    66:filename := 'conceptb.dss';
    67:filename := 'conceptc.dss';
    68:filename := 'conceptd.dss';
    69:filename := 'concepte.dss';
    70:filename := 'conceptf.dss';
    71:filename := 'conceptg.dss';
    72:filename := 'conceptth.dss';
    73:filename := 'concepti.dss';
    74:filename := 'conceptj.dss'
  end (specific case of filenbr)
end; (case filenbr member of concept file)
63: begin
  level := 1;
  Close(choicefile);
  Help(level);
  GoTo 2
end (case filenbr indicates ?)
end (case)

```

```

end; (ListConcepts)

overlay procedure Criteria(concept: Char20; personalfile: Char12;
                          Var quitindicator: integer);

(Criteria - governs the selection of the priority rankings of the
criteria. The first level of the AHP hierarchy for this
problem.
Object code is stored in overlay dss.002)

label 2;

Var userinput: file of userfile;
    inputfilerecord: userfile;
    answer: char;
    numcriteria, i: integer;
    comparisons: matrix;

begin
    Assign(userinput,personalfile);
    Reset(userinput);
    Seek(userinput,FileSize(userinput)-1);
    Read(userinput,inputfilerecord);
    numcriteria := 4;
    ClrScr;
    writeln;
    GetInfo(1,9); (Tells about attributes Block1)
    readln(answer);
    CheckYorN(answer);
    case answer of
        'Y','y': begin
            GetInfo(10,30); (Factors that impact attributes)
            writeln;
            write('      When finished reading, Please ');
            write('press RETURN: ');
            readln(answer)
        end;
        'N','n': ;
    end; (case of answer)
    ClrScr;
    for i := 1 to 10 do writeln;
    write('  You will now view the Comparison Scale.  Please');
    writeln(' read it');
    write('  carefully, as you will need it to ');
    writeln('make the comparisons. ');
    Delay(1500);
    GetInfo(30,53); (This is to display the AHP scale)
    write('      When finished reading, press RETURN: ');
    readln(answer);
    2:with inputfilerecord do
        begin
            CriteriaComparisons(comparisons,concept,
                                judgements.criteriachoices);
                                (Builds comparison matrix)
            AHP(comparisons, priorityvectors.criteriapriorities,

```

```

numcriteria, CRvector[1]);

ClrScr;
writeln;
writeln(b,'  PRIORITIZATION RESULTS FOR THE CRITERIA');
writeln;
writeln(b,'          ',concept);
writeln(b,'          Criteria          Priority-Weight');
writeln(b,'          -----          -----');
write(b,'          Performance          ');
writeln(priorityvectors.criteriapriorities[1]:8);
write(b,'          Schedule          ');
writeln(priorityvectors.criteriapriorities[2]:8);
write(b,'          Risk          ');
writeln(priorityvectors.criteriapriorities[3]:8);
write(b,'          Cost          ');
writeln(priorityvectors.criteriapriorities[4]:8);
writeln;
writeln(b,'          Consistency Ratio is: ',CRvector[1]:6);
conceptname := concept;
optiondata := filename
end; (with inputfilerecord)
writeln;
write('          The above is a priority vector based upon the ');
writeln('comparisons of the');
write('          criteria (attributes) that you have made. If');
writeln('the Consistency Ratio');
write('          value is above 1.0E-01 (0.10), signifying ');
writeln('inconsistent ');
write('          compairisons, you may want');
writeln('to repeat the compairson procedure. ');
write('          Is it acceptable (Y or N)? ');
readln(answer);
CheckYorN(answer);
if answer in ['N','n'] then
begin
writeln;
write(b,'Okay, we will go back and repeat the ');
writeln('comparisons. ');
Delay(1500);
GoTo 2
end; (if answer is No)
Seek(userinput,FileSize(userinput)-1);
Write(userinput,inputfilerecord);
writeln;
write('          You may Quit the DSS at this point and your ');
writeln('inputs up to this point');
write('          will be stored for later use. (Q or q and ');
writeln('RETURN to Quit, or');
write('          RETURN to continue: ');
readln(answer);
if answer in ['Q','q'] then Quit(quitindicator);
Close(userinput)
end; (Criteria)

```

overlay procedure PrioritizeOptions(personalfile: Char12;


```

        Var quitindicator, status: integer);

(PrioritizeOptions - governs the for "GET" vectors that find the
prioritized options based on each criteria.
Object code is stored in dss.002)

label 1,2,3,4,5;

Var userinput: file of userfile;
    inputfilerecord: userfile;
    hold: char;

begin
    Assign(userinput,personalfile);
    Reset(userinput);
    Seek(userinput,FileSize(userinput)-1);
    Read(userinput,inputfilerecord);
    case status of
        0: ;
        1: GoTo 2;
        2: GoTo 3;
        3: GoTo 4;
        4: GoTo 5
    end;
    GetInfo(64,74);
    readln(hold);
2:GetPerformanceVector(inputfilerecord,quitindicator);
    if quitindicator = 1 then GoTo 1;
3:GetScheduleVector(inputfilerecord,quitindicator);
    if quitindicator = 1 then GoTo 1;
4:GetRiskVector(inputfilerecord,quitindicator);
    if quitindicator = 1 then GoTo 1;
5:GetCostVector(inputfilerecord,quitindicator);
1:Seek(userinput,FileSize(userinput)-1);
    Write(userinput,inputfilerecord);
    Close(userinput)
end; (PrioritizeOptions)

overlay procedure FinalOptionVector(Var quitindicator,
        status: integer; personalfile: Char12);

(FinalOptionVector - synthesizes the AHP results from the criteria
and for the options based on the attributes.
Object code is stored in dss.002)

label 1;

Var userinput: file of userfile;
    inputfilerecord: userfile;
    i,,numoptions: integer;
    answer: char;

begin
    status := 10;
    if quitindicator = 1 then GoTo 1;

```

```

numoptions := 0;
Assign(userinput, personalfile);
Reset(userinput);
Seek(userinput, FileSize(userinput)-1);
Read(userinput, inputfilerecord);
for i := 1 to maxnumoptions do if
    inputfilerecord.options[i] <> ' ' then
        numoptions := numoptions + 1;
GetInfo(149,156);
with inputfilerecord do
begin
    with priorityvectors do
    begin
        writeln;
        writeln(b, '                FINAL PRIORITIES');
        writeln(b, '                -----');
        writeln(b, '                ', conceptname);
        writeln(b, '                Option                Priority');
        writeln(b, '                -----                -----');
        for i := 1 to numoptions do
            begin
                finalpriorities[i] := ((criteriapriorities[1] *
                    performancevector[i]) + (criteriapriorities[2] *
                    schedulevector[i]) + (criteriapriorities[3] *
                    riskvector[i]) + (criteriapriorities[4] *
                    costvector[i]));
                write('                ', i, ' ', options[i]:20, ' ');
                writeln('                ', finalpriorities[i]:8)
            end
        end (with priorityvectors do)
    end; (with inputfilerecord do)
    writeln;
    write('                Does this final vector make sense to you ');
    write('(Y or N)? ');
    readln(answer);
    CheckYorN(answer);
    if answer in ['N', 'n'] then
        begin
            writeln;
            write('                Do you wish to erase current effort and ');
            write('start over (Y or N)? ');
            readln(answer);
            CheckYorN(answer);
            if (answer in ['Y', 'y']) then
                begin
                    quitindicator := 0;
                    status := 7;
                    Close(userinput);
                    InitializeUserFile(personalfile, status);
                    GoTo 1
                end (Yes, repeat the process)
            end; (if answer is no, finalvector does not make sense)
        end
    Seek(userinput, FileSize(userinput)-1);
    Write(userinput, inputfilerecord);
    Close(userinput);

```

```
1:writeln  
  end;      (FinalOptionVector)
```

```

*****
*****      Menus and tables stored on disk and accessed      *****
*****      by the GetInfo procedure.                          *****
*****

```

At this time, you will make a series of pairwise comparisons between the attributes (criteria) - Performance, Schedule, Risk, and Cost. The comparisons will help define which attribute you consider most important.

Do you wish to see a further definition of the attributes?
(Y or y for Yes, N or n for No)

Factors that impact the attributes are as follows:

Performance

Survivability
Coverage
Capacity
Quality
Reliability
Timeliness
Availability

Cost

R & D
Replacement
Deployment
Resupply

Schedule

Earliest Completion
Date
Earliest production
date
Potential schedule
variability

Risk

Number of High Risk
Tech Issues
Number of proven
technologies
Number of technologies
common to other concepts

COMPARISON SCALE

INTENSITY OF IMPORTANCE

DEFINITION

EXPLANATION

1	Equal importance	Two criteria contribute equally to the objective.
3	Weak importance of one over the other	Slightly favor one criterion over the other.
5	Essential or strong importance of one	Strongly favor one criterion over another.
7	Very strong or demonstrated performance	A criterion has demonstrated its dominance.
9	Absolute importance	Evidence favoring one criterion is of highest

order.

2,4,6,8

Intermediate values

Compromise as needed.

Example - Schedule vs Cost: C7

The above example shows the comparison procedure. When a comparison is presented to you, input the first letter of the attribute that dominates the other, and then the integer value of how much it dominates, from the comparison scale. Therefore, the above example shows that Cost has very strong dominance over Schedule (example only).

To continue, press RETURN.

You will compare the options of the chosen concept at this time. The general procedure will be to compare the options with respect to each of the four criteria individually. Once this is accomplished the four resulting vectors can be combined and multiplied with the criteria vector you have just completed, providing a final ordering of the options, based upon your judgements.

To continue, press RETURN

The performance values for the available options will be displayed at this time. You will then perform a series of pairwise comparisons that will define which option you feel performs "best".

Do you wish to review the Comparison Scale (Y or N)?

The performance values are based on the following scale:

INTENSITY	EXPLANATION
5	Option strongly exceeds performance requirement
3	Option meets performance requirement
1	Option has serious shortcomings in performance
2,4	Compromise as needed

Example:	Option A	Option B
Coverage	2	5
Capacity	3	2

For this series of comparisons, the procedure will be to input the number of the option that dominates the comparison and then the comparison value.

Ex: Option 1 vs Option 2: 27 - Option 2 dominates strongly

The options will now be ordered according to the schedule criteria. The procedure will be exactly the same as for

the performance criteria. The number of years until each Concept Option is displayed, along with the mean value for each option. Then, you will be asked to compare the options based upon this criteria. Again, the format will be the number of the option, then the value from the comparison scale. Ex: 17 - means option 1 dominates the comparison with a value of 7 (strong domination).

The options will now be ordered according to the third criteria - Risk. The procedure will be the same as for the first two criteria. The displays available will be a display of the options and the mean risk from their technology issues, or a detailed list of the risk associated with each technology issue under a given option. Format for the comparisons will again be two integers, i.e. 17.

The Risk values are based upon the following scale:

Level -----	Risk ----	Explanation -----
1	very low	High probability that Tech issue can be solved in time to meet IOC given current effort.
2	low	Probability is high but but some doubt exists.
3	medium	Probability of completion is 50/50.
4	high	Low probability of solving issue.
5	very high	Very strong possibility that Tech issue is unsolvable in necessary time.

The options will now be ordered according to the Cost criteria. Again, the procedure will be the same as for the previous attributes. The system will display each option with its associated total cost to solve all Technology Issues unique to the given concept option. The pairwise comparison procedure will follow the same format.

Ex: Option 1 vs Option 2: 17 - Option 1 dominates Option 2 strongly in this example.

The following vector will be the priorities of the options for this concept based upon your evaluations of the criteria, performance, schedule, risk and cost, and the different options compared in relation to these attributes. It should be noted that this priority vector is subjective, as it is based upon your judgement.

WELCOME TO THE A.F. SPACE TECHNOLOGY
CENTER DECISION SUPPORT SYSTEM (DSS)

This DSS is designed to take you through a series of decision steps that will help you to prioritize the concept options that are contained in a concept. The system uses a method called the Analytic Hierarchy Process, which was developed by Thomas L. Saaty to structure complex decision processes. By following the outlined instructions, you will place your subjective judgements into a series of matrices that are solved for the final priority of the listed options.

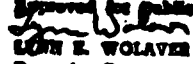
As a final note, this software is currently configured for a Zenith Z-100 computer running under the MS-DOS operating system. If this computer is a Z-100, the key sequence <shift F12> will print the screen at any time in the program. That is, if you have a printer, and it is hooked up, and it is turned on.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

A172379

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GOS/OS/85D-17			5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/ENS		7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB OH 45433			7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Space Technology Center		8b. OFFICE SYMBOL (If applicable) AFSTC/YHP		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code) Kirtland AFB NM 87117			10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See Box 19			PROGRAM ELEMENT NO.	TASK NO.
12. PERSONAL AUTHOR(S) Bruce G. Schinelli, B.S., Capt, USAF			PROJECT NO.	WORK UNIT NO.
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Yr., Mo., Day) 85 Dec 13
				15. PAGE COUNT 170
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB. GR.		
05	10		Decision Support Systems, Microcomputers	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
Title: A Decision Support System for Space Technology Tradeoffs: A Microcomputer Application				
Thesis advisor: Mark M. Mekaru, Lt Col, PhD, USAF				
<div style="text-align: right;"> <p>Approved for public release LAW 872 100-4  LON E. WOLAVER Dean for Research and Professional Development Air Force Institute of Technology (AFIT) Wright-Patterson AFB OH 45433</p> </div>				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Mark M. Mekaru, Lt Col, USAF		22b. TELEPHONE NUMBER (Include Area Code) (513) 255-3362		22c. OFFICE SYMBOL AFIT/ENS

END

1-56

DTIC